



UNIVERSITÀ DEGLI STUDI DI BERGAMO

Scuola di Ingegneria

Corso di laurea in Ingegneria informatica

Classe n. L-8 Ingegneria dell'informazione

Il problema della brachistocrona: analisi teorica e sviluppo di una simulazione software della caduta di gravi lungo curve

Candidato:

Marco Locatelli

Matricola n. 1079236

Relatrice:

Chiar.ma Prof.ssa

Giulia Maria Dalia Furioli

Anno Accademico 2023/2024

Abstract

Il problema della brachistocrona consiste nel determinare la curva lungo cui una massa puntiforme, soggetta alla forza di gravità, discende dal punto di partenza a quello di arrivo nel minor tempo possibile.

Dopo un'introduzione al problema e una presentazione di diversi approcci risolutivi, viene sviluppata una simulazione software in linguaggio Java, che permette di visualizzare e confrontare la caduta di gravi lungo diverse curve.

Il progetto si fonda su un preliminare studio teorico, seguito dalla progettazione e dall'implementazione del sistema software, secondo i metodi dell'ingegneria.

Indice

1. Introduzione.....	5
2. Introduzione storica	6
2.1. La sfida di Johann Bernoulli	6
2.2. Le radici storiche: Galileo e il moto accelerato	6
2.3. Le soluzioni e la loro importanza.....	7
2.3.1. Johann Bernoulli e il principio di Fermat	7
2.3.2. Isaac Newton e la geometria sintetica	7
2.3.3. Leibniz e il calcolo infinitesimale.....	7
2.3.4. Jakob Bernoulli e il calcolo delle variazioni.....	8
3. Introduzione al problema	8
3.1. Enunciato	8
3.2. Restrizione al piano verticale.....	8
3.3. Modello del problema.....	10
3.4. Osservazioni generali.....	10
4. Soluzione personale.....	12
4.1. Analisi della caduta lungo piano inclinato.....	12
4.2. Approccio con una spezzata a due segmenti.....	13
4.3. Minimizzazione del tempo di caduta	15
4.4. Generalizzazione a una spezzata con più segmenti	18
4.5. Condizione della brachistocrona.....	19
4.6. Risoluzione equazione differenziale	20
5. Cicloide.....	23
5.1. Parametrizzazione	24
5.2. Proprietà.....	25
5.3. Equazione cartesiana.....	26
5.4. Verifica che la cicloide è la curva brachistocrona	28
5.5. Determinazione del ramo di cicloide che rappresenta la curva brachistocrona	29
6. Soluzione con calcolo delle variazioni.....	47
6.1. Modello dinamico e tempo di caduta.....	47
6.2. Formulazione variazionale del problema.....	48
6.3. Equazione di Eulero-Lagrange e riduzione di Beltrami.....	49
6.4. Risoluzione dell'equazione differenziale.....	49
6.5. Condizioni al bordo	50

6.6.	Studio della minimalità della soluzione con l'analisi della convessità della lagrangiana	51
7.	Soluzione con teoria del controllo ottimo	55
7.1.	Definizione del sistema dinamico.....	55
7.2.	Hamiltoniana e principio del massimo di Pontryagin.....	56
7.3.	Condizioni per il controllo ottimo.....	57
7.4.	Equazioni del co-stato.....	58
7.5.	Determinazione della curva ottimale	59
7.6.	Verifica della soluzione	61
8.	Simulazione software caduta di gravi lungo curve	63
8.1.	Paradigma di programmazione e strumenti utilizzati	64
8.2.	Modello di processo di sviluppo del software	65
8.3.	Requisiti	66
8.4.	Architettura del software	67
8.5.	Progettazione	68
8.6.	Design pattern	73
8.7.	Implementazione.....	78
8.7.1.	Curve	79
8.7.2.	Parabola.....	81
8.7.3.	Circonferenza	86
8.7.4.	Cicloide	101
8.7.5.	Spline cubica.....	107
8.7.6.	Simulazione	113
8.8.	Testing	117
8.9.	Manutenzione.....	117
8.10.	Demo.....	118
8.11.	Sviluppi futuri	122
9.	Conclusioni.....	122
	Riferimenti bibliografici	125

1. Introduzione

Pochi problemi nella storia della matematica possiedono il fascino e il prestigio del problema della brachistocrona. Tale fama è dovuta a molteplici ragioni:

- Il problema fu proposto in origine come una sfida intellettuale volta a mettere alla prova le abilità matematiche dei più grandi pensatori dell'epoca. Sfida che fu accolta da molte menti illustri e vide contrapporsi giganti della scienza come Newton, Leibniz e i fratelli Bernoulli. Ciò ne evidenzia la grande rilevanza storica.
- Si contraddistingue per una ricca e affascinante aneddotta, che ha contribuito ad accrescerne la fama e a renderlo un simbolo emblematico della storia della matematica.
- Per una ragione di estetica matematica. La soluzione del problema, la cicloide, risulta inaspettata e sorprendente, poiché nulla sembrerebbe suggerirla intuitivamente. Nonostante la formulazione del problema sia concisa, semplice e comprensibile da chiunque, risulta molto difficile immaginare anche solo come iniziare ad affrontarlo. È necessario invece elaborare un sofisticato e rigoroso ragionamento matematico, che conduce inequivocabilmente all'unica soluzione, da cui traspare un ideale di eleganza e perfezione matematica.
- Il motivo principale dell'enorme importanza del problema è l'impulso che diede allo sviluppo di nuove idee e tecniche, che confluiirono nella teoria del calcolo delle variazioni, un campo dell'analisi matematica fondamentale per la fisica e l'ingegneria. Il problema rese evidente la centralità dell'ottimizzazione nello studio dei fenomeni e pose le basi per lo sviluppo di quello che può essere considerato il principio più profondo e fondamentale di tutta la fisica: il principio di minima azione, che funge da filo conduttore delle principali teorie, accomunando meccanica classica, elettromagnetismo, relatività generale e meccanica quantistica. Oltre alla fisica, la sua influenza si estende fortemente anche in ingegneria, ad esempio in robotica, logistica, automatica e teoria dei sistemi dinamici, nello specifico con la teoria del controllo ottimo. Il principio rappresenta una delle idee più potenti e generali mai sviluppate dall'uomo, poiché trova applicazione trasversale in moltissime aree della scienza, fornendo un quadro unificato per comprendere l'evoluzione dei sistemi dinamici.

2. Introduzione storica

2.1. La sfida di Johann Bernoulli

Johann Bernoulli, un matematico svizzero professore all'università di Groninga, nel Giugno del 1696, pubblicò una sfida intellettuale sulla rivista scientifica *"Acta Eruditorum"*, invitando i più grandi matematici dell'epoca a risolvere il “*problema novum.*” [1]

“Johann Bernoulli si rivolge ai matematici più brillanti del mondo. Nulla è più attraente per le persone intelligenti di un problema onesto e stimolante, la cui possibile soluzione conferirà fama e rimarrà come un monumento duraturo. Seguendo l'esempio di Pascal, Fermat, ecc., spero di guadagnarmi la gratitudine dell'intera comunità scientifica presentando ai migliori matematici del nostro tempo un problema che metterà alla prova i loro metodi e la forza del loro intelletto. Se qualcuno mi comunicherà la soluzione del problema proposto, lo proclamerò pubblicamente degno di lode.” [2]

“Se in un piano verticale sono dati due punti A e B, si richiede di specificare l'orbita AMB del punto mobile M, lungo la quale, partendo da A e sotto l'influenza del proprio peso, esso raggiunge B nel minor tempo possibile. Affinché coloro che sono appassionati di tali questioni siano stimolati a risolvere questo problema, è bene sapere che esso non è, come potrebbe sembrare, puramente speculativo e privo di utilità pratica. Al contrario, sembra addirittura – e questo potrebbe essere difficile da credere – che sia molto utile anche per altre discipline oltre alla meccanica. Per evitare conclusioni affrettate, si deve osservare che la retta è certamente la linea di distanza più breve tra A e B, ma non è quella che viene percorsa nel minor tempo. Tuttavia, la curva AMB – che rivelerò se entro la fine di quest'anno nessun altro l'avrà trovata – è ben nota a coloro che si occupano di geometria.” [2]

2.2. Le radici storiche: Galileo e il moto accelerato

Prima di Bernoulli, Galileo Galilei aveva compiuto studi sul moto accelerato lungo piani inclinati, gettando le basi per comprendere i principi fisici alla base del problema della brachistocrona. Nel suo *"Dialogo sopra i due massimi sistemi del mondo"* (1638), Galileo osservò che il tempo di discesa lungo un piano inclinato dipende dalla lunghezza del piano

stesso e dall'altezza relativa, ma concluse erroneamente che l'arco di cerchio fosse la curva ottimale per il tempo minimo. Questa intuizione era scorretta, ma rappresentò un primo tentativo significativo di affrontare il problema.

2.3. Le soluzioni e la loro importanza

La sfida attirò l'attenzione delle migliori menti matematiche dell'epoca. Le soluzioni arrivarono da figure illustri come Jakob Bernoulli, Gottfried Wilhelm Leibniz, il marchese de l'Hôpital e Isaac Newton.

2.3.1. Johann Bernoulli e il principio di Fermat

La soluzione di Johann Bernoulli si basava sul principio di Fermat del tempo minimo, già applicato da Huygens nell'ottica per spiegare la rifrazione della luce. Bernoulli immaginò il moto del corpo lungo la curva come analogo al percorso di un raggio di luce che attraversa un mezzo con densità variabile. Usò il calcolo differenziale per derivare la curva e dimostrò che la brachistocrona era un arco di cicloide, la curva descritta da un punto su una circonferenza che rotola senza slittare su una linea retta.

2.3.2. Isaac Newton e la geometria sintetica

Isaac Newton inviò una soluzione anonima, redatta in termini geometrici piuttosto che analitici. Bernoulli, riconoscendo immediatamente lo stile e l'autorità della risposta, esclamò: "tanquam ex ungue leonem" [2] ("riconosco il leone dalla zampata"). Newton utilizzò la geometria sintetica per derivare la cicloide, una soluzione che evidenziava la sua abilità.

2.3.3. Leibniz e il calcolo infinitesimale

Gottfried Wilhelm Leibniz, che all'epoca era coinvolto in una disputa con Newton sulla paternità del calcolo, rispose alla sfida enfatizzando l'uso del suo calcolo infinitesimale. Sostenne che il problema poteva essere risolto solo grazie a questa nuova disciplina, esaltandone le potenzialità applicative. La sua soluzione fu fondamentale per consolidare l'uso del calcolo infinitesimale nella risoluzione di problemi fisici.

2.3.4. Jakob Bernoulli e il calcolo delle variazioni

Il fratello di Johann, Jakob Bernoulli, propose una soluzione che introdusse tecniche generalizzabili a una vasta gamma di problemi di ottimizzazione. Il suo approccio anticipava il moderno calcolo delle variazioni, un campo che avrebbe trovato applicazioni in fisica, ingegneria e scienze economiche.

3. Introduzione al problema

3.1. Enunciato

Determinare la curva lungo la quale una massa puntiforme, partendo da un punto A e muovendosi sotto l'azione della sola forza di gravità, raggiunge un punto B (posto a una quota non superiore ad A) nel minor tempo possibile. Si assume che la massa inizi il moto da ferma e che non vi siano forze dissipative come l'attrito.

3.2. Restrizione al piano verticale

La gravità agisce lungo la direzione verticale y . I due punti A e B e la direzione dell'accelerazione di gravità individuano un piano verticale xy . La velocità della massa puntiforme, in assenza di attrito, dipende unicamente dalla variazione di energia potenziale, cioè dalla quota verticale y . Si considera una curva qualsiasi γ nello spazio xyz avente parametrizzazione:

$$\gamma = \begin{cases} x = x(t) \\ y = t \\ z = z(t) \end{cases} .$$

Si considera la sua proiezione ortogonale $\bar{\gamma}$ sul piano verticale xy contenete A e B, la cui parametrizzazione è quindi:

$$\bar{\gamma} = \begin{cases} x = x(t) \\ y = t \\ z = 0 \end{cases} .$$

γ e $\bar{\gamma}$ condividono i punti A e B di partenza e arrivo.

A parità di quota verticale y (stessa energia potenziale), la velocità è la stessa per entrambe le curve.

Per calcolare il tempo T necessario a percorrere le due curve γ e $\bar{\gamma}$ si scrive:

$$v = \frac{ds}{dt} \rightarrow dt = \frac{ds}{v}.$$

Da cui:

$$T[\gamma] = \int_{\gamma} \frac{ds}{v}. \quad T[\bar{\gamma}] = \int_{\bar{\gamma}} \frac{ds}{v}.$$

$$\text{Su } \gamma: ds = \sqrt{x'(t)^2 + 1 + z'(t)^2} \cdot dt. \quad \text{Su } \bar{\gamma}: ds = \sqrt{x'(t)^2 + 1 + 0} \cdot dt.$$

Quindi i tempi si possono calcolare come:

$$T[\gamma] = \int_{y(A)}^{y(B)} \frac{\sqrt{x'(t)^2 + 1 + z'(t)^2}}{v(y)} dt. \quad T[\bar{\gamma}] = \int_{y(A)}^{y(B)} \frac{\sqrt{x'(t)^2 + 1}}{v(y)} dt.$$

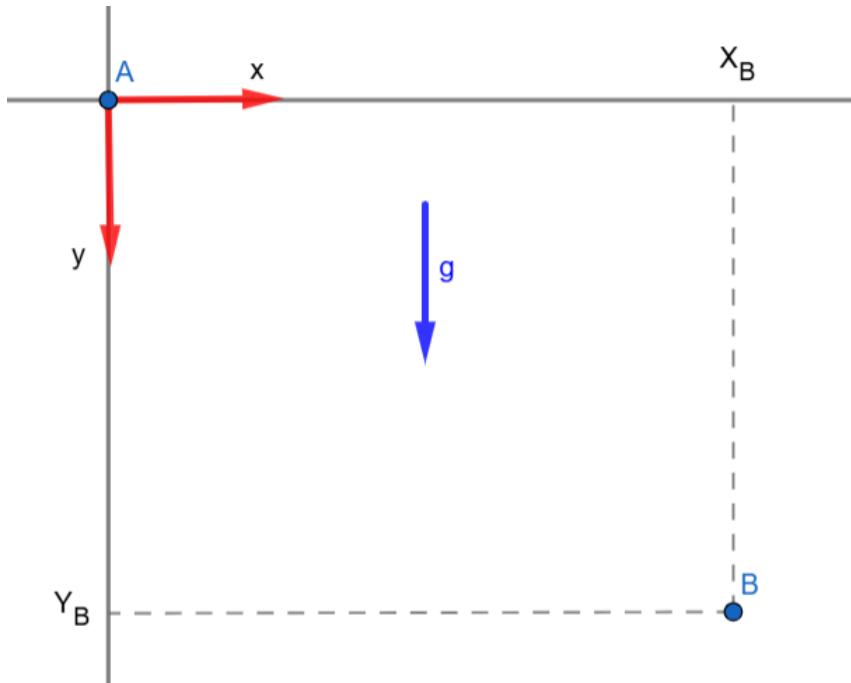
$v(y)$ è la stessa per $T[\gamma]$ e per $T[\bar{\gamma}]$.

Poiché $\sqrt{x'(t)^2 + 1 + z'(t)^2} \geq \sqrt{x'(t)^2 + 1}$ si deduce che $T[\gamma] \geq T[\bar{\gamma}]$.

Data qualsiasi curva nello spazio dal punto A al punto B, la sua proiezione ortogonale sul piano verticale passante per A e B fornisce quindi un tempo di caduta inferiore.

Dunque, qualsiasi deviazione della curva di caduta al di fuori del piano verticale non fornisce alcun vantaggio, allunga solo la traiettoria, peggiorando il tempo totale di percorrenza. Di conseguenza, la curva brachistocrona si colloca necessariamente nel piano verticale passante per i punti A e B. Ci si può quindi limitare ad analizzare tale piano nella ricerca della soluzione del problema.

3.3. Modello del problema



Il punto di partenza A è posizionato nell' origine del sistema di riferimento. La massa parte da ferma, pertanto la velocità in A è nulla: $v_A = 0$.

Il punto di arrivo B ha una posizione generica inferiore ad A, di coordinate (X_B, Y_B) .

Per semplificare l'uso dei segni si stabilisce la direzione positiva dell'asse y verso il basso, poiché tale è la direzione dell'accelerazione di gravità g . Di conseguenza $Y_B > 0$.

3.4. Osservazioni generali

Si può fare la seguente assunzione: la curva ricercata deve essere una curva continua poiché si sta modellando la caduta di punti materiali che si muovono lungo traiettorie continue. Se la curva non fosse continua, i punti materiali non ne seguirebbero la traiettoria nei punti di discontinuità. Si assume anche che la curva brachistocrona sia derivabile, affinché la velocità della massa sia ben definita in ogni punto.

Inoltre, la curva deve essere una funzione di x perché la componente x della velocità del punto materiale non può invertire direzione, poiché l'accelerazione gravitazionale agisce solamente lungo la direzione y . Se la curva continua non fosse una funzione di x , avrebbe un punto di estremo locale rispetto a x , in corrispondenza del quale la massa non

seguirebbe la traiettoria della curva, ma proseguirebbe nel suo moto dovuto alla componente della velocità già acquisita nella direzione di x .

Pertanto, la soluzione deve essere una funzione $C^1[0, X_B]$ di x .

Si considera $y = 0$ come quota di riferimento in cui l'energia potenziale è nulla. Inoltre, poiché si ha scelto il verso positivo dell'asse y verso il basso, dunque concorde al verso dell'accelerazione di gravità, bisogna aggiungere un segno meno all'espressione dell'energia potenziale, per garantire che il suo gradiente rimanga comunque opposto alla forza gravitazionale, come stabilito dalla relazione $F = -\nabla U$, per le forze conservative.

L'energia potenziale si esprime dunque come: $U = -mgy$.

È possibile calcolare l'energia meccanica E_A nel punto di partenza A:

$$E_A = K_A + U_A = \frac{1}{2}mv_A^2 - mgy_A = 0.$$

Dove K_A denota l'energia cinetica in A e U_A l'energia potenziale in A.

Poiché l'unica forza che agisce, la gravità, è conservativa, si ha conservazione dell'energia meccanica:

$$E = 0 \rightarrow \frac{1}{2}mv^2 - mgy = 0 \rightarrow v^2 = 2gy \rightarrow v = \sqrt{2gy}, \forall y \geq 0.$$

Il modulo della velocità dipende solamente dalla quota y della massa.

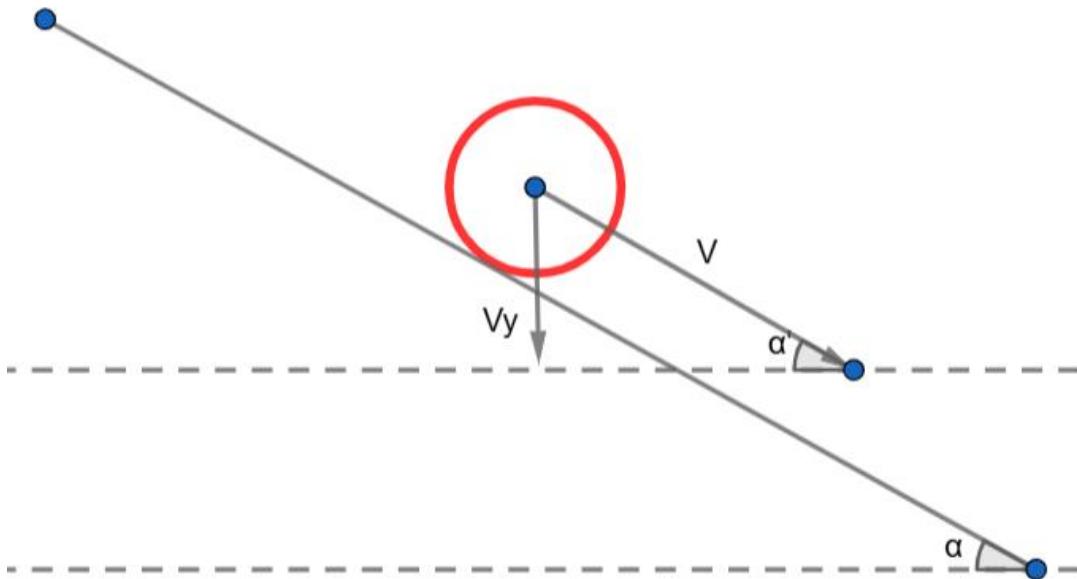
4. Soluzione personale

Nell'estate 2021 mi sono messo alla prova con l'obiettivo di risolvere il problema autonomamente, facendo uso esclusivamente dei concetti matematici da me conosciuti al tempo: derivate, integrali ed equazioni differenziali.

4.1. Analisi della caduta lungo piano inclinato

Si comincia considerando la caduta di una massa lungo un piano inclinato a inclinazione costante α ($0 \leq \alpha \leq \frac{\pi}{2}$).

La componente verticale della velocità è: $v_Y = v \cdot \sin \alpha = \sqrt{2gy} \cdot \sin \alpha$.



Il tempo totale T per percorrere lo spazio verticale da y_i a y_f , con $y_f > y_i$, lungo una traiettoria generica, conoscendo la componente y della velocità in funzione della quota y , è dato da:

$$v_Y = \frac{dy}{dt} \rightarrow dt = \frac{1}{v_Y} dy \rightarrow \int_0^T dt = \int_{y_i}^{y_f} \frac{1}{v_Y} dy \rightarrow T = \int_{y_i}^{y_f} \frac{1}{v_Y} dy.$$

La velocità rimane sempre riferita alla quota $y = 0$ come punto in cui l'energia cinetica è nulla e, quindi, la sua espressione rimane comunque $v(y) = \sqrt{2gy}$. Ciò significa che si sta considerando che la massa abbia velocità iniziale $v(y_i) = \sqrt{2gy_i}$.

Per un segmento inclinato a pendenza α costante è quindi :

$$\begin{aligned} T &= \int_{y_i}^{y_f} \frac{1}{\sqrt{2gy \cdot \sin \alpha}} dy = \frac{1}{\sin \alpha} \cdot \int_{y_i}^{y_f} \frac{1}{\sqrt{2gy}} dy = \frac{1}{\sin \alpha} \cdot \left(\sqrt{\frac{2y_f}{g}} - \sqrt{\frac{2y_i}{g}} \right) = \\ &= \sqrt{\frac{2}{g}} \cdot \frac{1}{\sin \alpha} \cdot (\sqrt{y_f} - \sqrt{y_i}). \end{aligned}$$

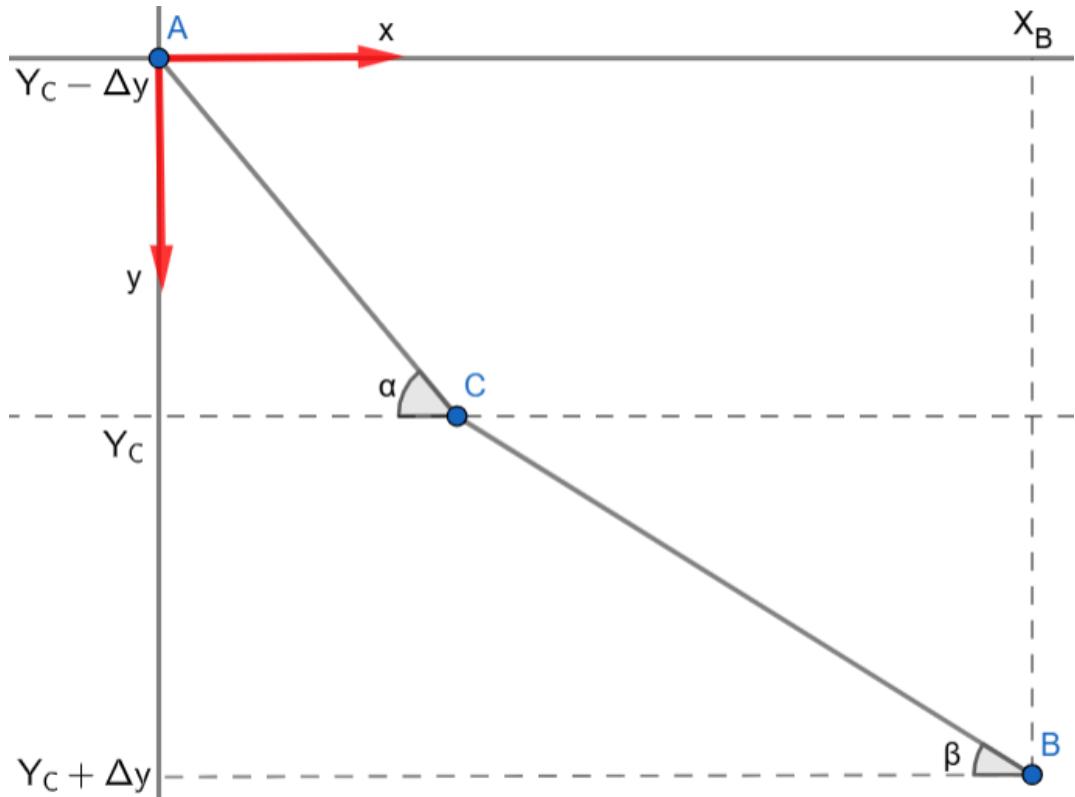
4.2. Approccio con una spezzata a due segmenti

Si cerca a questo punto di semplificare il problema in modo da avere il tempo di caduta funzione solamente di una variabile, per riuscire a minimizzarlo con il calcolo differenziale. Per fare ciò si considera una spezzata di due segmenti che collegano i punti A e B. Il vertice comune C della spezzata si stabilisce che sia posizionato lungo la direzione y a metà tra A e B. Il vertice C può variare lungo la retta orizzontale $y = Y_C$. Per rendere il risultato generale e riutilizzabile si considera che la massa possa avere una velocità iniziale e cioè che A abbia una coordinata Y_A possibilmente maggiore di 0.

Poiché C è a metà tra A e B lungo y, le ordinate di A e B possono essere indicate con $Y_C - \Delta y$ e $Y_C + \Delta y$ rispettivamente. Con Δy si indica la differenza lungo y tra C - A e B - C, e rappresenta una quantità positiva che verrà successivamente fatta tendere a 0.

Si vuole a questo punto trovare la spezzata che minimizza il tempo di caduta da A a B. Ciò è possibile con il calcolo differenziale, poiché il tempo di caduta lungo tutte le possibili spezzate si può esprimere come funzione di una sola variabile. Come tale variabile si può usare l'angolo α , che rappresenta l'inclinazione del primo segmento AC della spezzata rispetto all'orizzontale.

L'analisi del tempo di caduta per una coppia di segmenti fornisce una condizione di minimizzazione del tempo che, applicata localmente a ogni tratto infinitesimo del percorso, conduce all'identificazione della curva brachistocrona.



Per la spezzata ACB il tempo totale di caduta da a A a B è :

$$\begin{aligned}
 T &= \sqrt{\frac{2}{g}} \cdot \frac{1}{\sin \alpha} \cdot (\sqrt{y_c} - \sqrt{y_c - \Delta y}) + \sqrt{\frac{2}{g}} \cdot \frac{1}{\sin \beta} \cdot (\sqrt{y_c + \Delta y} - \sqrt{y_c}) = \\
 &= \sqrt{\frac{2}{g}} \cdot \left(\frac{(\sqrt{y_c} - \sqrt{y_c - \Delta y})}{\sin \alpha} + \frac{(\sqrt{y_c + \Delta y} - \sqrt{y_c})}{\sin \beta} \right).
 \end{aligned}$$

L'angolo β si può esprimere in funzione di α imponendo la seguente condizione:

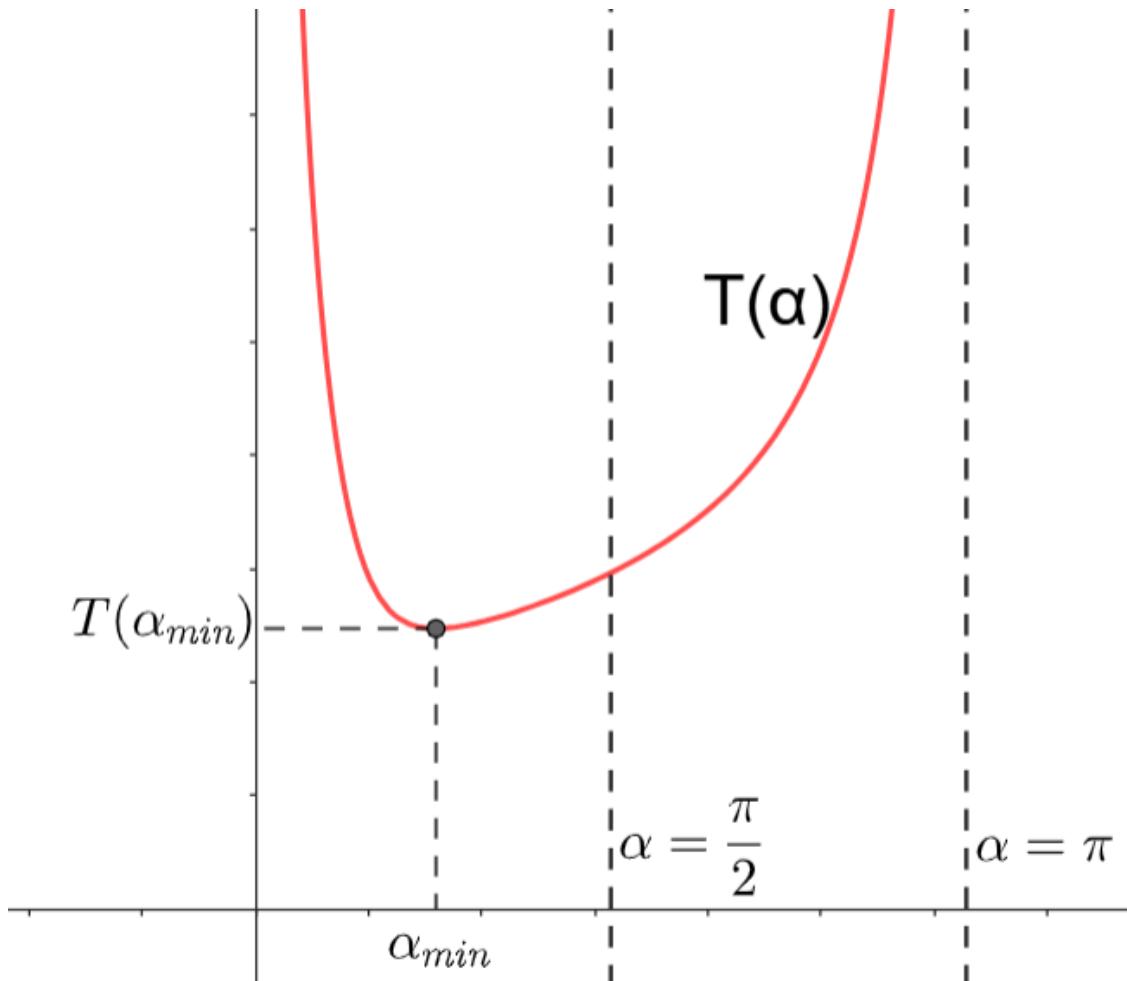
$$X_B = dy \cdot \cot \alpha + dy \cdot \cot \beta \rightarrow \cot \beta = \frac{X_B}{dy} - \cot \alpha.$$

Ponendo $k = \frac{X_B}{dy}$ si ottiene $\cot \beta = k - \cot \alpha$.

Per l'uguaglianza trigonometrica $\frac{1}{\sin \alpha} = \sqrt{1 + (\cot \alpha)^2}$ si giunge a poter esprimere T in funzione solo di α , cioè:

$$T(\alpha) = \sqrt{\frac{2}{g}} \cdot \left[(\sqrt{y_c} - \sqrt{y_c - \Delta y}) \cdot \sqrt{1 + (\cot \alpha)^2} + (\sqrt{y_c + \Delta y} - \sqrt{y_c}) \cdot \sqrt{1 + (k - \cot \alpha)^2} \right].$$

La funzione è periodica e il primo periodo è definito per $0 < \alpha < \pi$. Tuttavia, il dominio significativo per il problema è $0 < \alpha < \frac{\pi}{2}$.

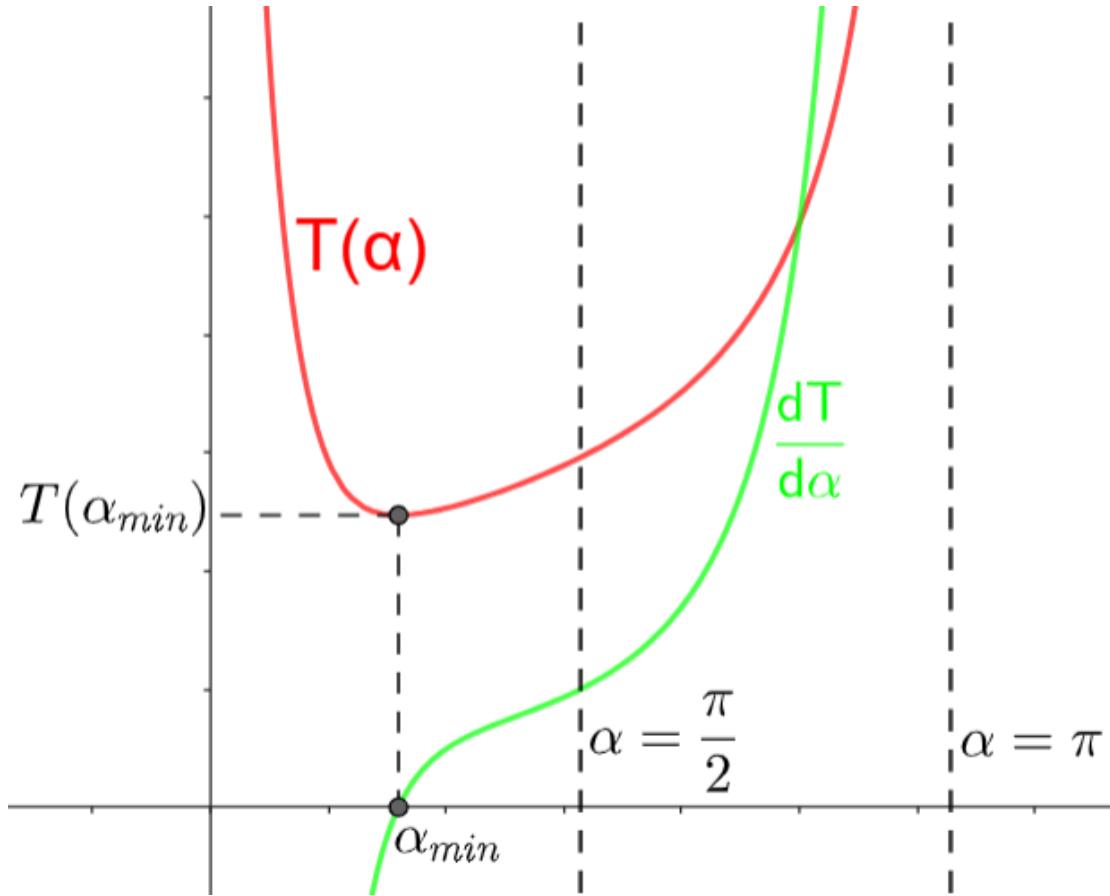


La funzione ha un asintoto verticale per $\alpha = 0$, poiché in tal caso il segmento sarebbe orizzontale e la massa non inizierebbe a scivolare. Inoltre, presenta un punto di minimo assoluto per un certo α nel dominio.

4.3. Minimizzazione del tempo di caduta

Si calcola la derivata prima di T rispetto ad α

$$\frac{dT}{d\alpha} = \sqrt{\frac{2}{g}} \cdot \left(-\frac{\cot \alpha \cdot (1 + (\cot \alpha)^2)}{\sqrt{1 + (\cot \alpha)^2}} \cdot (\sqrt{y_c} - \sqrt{y_c - \Delta y}) + \frac{(k - \cot \alpha) \cdot (1 + (\cot \alpha)^2)}{\sqrt{1 + (k - \cot \alpha)^2}} \cdot (\sqrt{y_c + \Delta y} - \sqrt{y_c}) \right).$$



Per trovare il valore di α che minimizza il tempo di caduta si pone la derivata a 0, cioè:

$$\sqrt{\frac{2}{g}} \cdot \left(-\frac{\cot \alpha \cdot (1 + (\cot \alpha)^2)}{\sqrt{1 + (\cot \alpha)^2}} \cdot (\sqrt{y_c} - \sqrt{y_c - \Delta y}) + \frac{(k - \cot \alpha) \cdot (1 + (\cot \alpha)^2)}{\sqrt{1 + (k - \cot \alpha)^2}} \cdot (\sqrt{y_c + \Delta y} - \sqrt{y_c}) \right) = 0.$$

Semplificando $\sqrt{\frac{2}{g}}$ e $(1 + (\cot \alpha)^2)$ si ottiene :

$$-\frac{\cot \alpha}{\sqrt{1 + (\cot \alpha)^2}} \cdot (\sqrt{y_c} - \sqrt{y_c - \Delta y}) + \frac{(k - \cot \alpha)}{\sqrt{1 + (k - \cot \alpha)^2}} \cdot (\sqrt{y_c + \Delta y} - \sqrt{y_c}) = 0$$

$$\text{e quindi : } \frac{\cot \alpha}{\sqrt{1 + (\cot \alpha)^2}} \cdot (\sqrt{y_c} - \sqrt{y_c - \Delta y}) = \frac{(k - \cot \alpha)}{\sqrt{1 + (k - \cot \alpha)^2}} \cdot (\sqrt{y_c + \Delta y} - \sqrt{y_c}).$$

A questo punto si potrebbe pensare di voler trovare l'esatta espressione per α che rende minimo $T(\alpha)$. Tuttavia, dipenderebbe dal parametro k , e quindi dalla specifica posizione di B rispetto ad A. Risulta invece più utile ottenere una relazione generale che deve valere per qualsiasi spezzata a tempo minimo, indipendentemente da k .

Si risostituisce quindi $\cot \beta = k - \cot \alpha$, ottenuta in precedenza:

$$\frac{\cot \alpha}{\sqrt{1 + (\cot \alpha)^2}} \cdot (\sqrt{y_c} - \sqrt{y_c - \Delta y}) = \frac{\cot \beta}{\sqrt{1 + (\cot \beta)^2}} \cdot (\sqrt{y_c + \Delta y} - \sqrt{y_c}).$$

Dall'uguaglianza trigonometrica $\cos \alpha = \frac{\cot \alpha}{\sqrt{1 + (\cot \alpha)^2}}$ si deduce

$$\cos \alpha \cdot (\sqrt{y_c} - \sqrt{y_c - \Delta y}) = \cos \beta \cdot (\sqrt{y_c + \Delta y} - \sqrt{y_c}).$$

Si riformulano le differenze tra radicali moltiplicando e dividendo per il coniugato:

$$\cos \alpha \cdot \frac{\Delta y}{\sqrt{y_c} + \sqrt{y_c - \Delta y}} = \cos \beta \cdot \frac{\Delta y}{\sqrt{y_c + \Delta y} + \sqrt{y_c}},$$

dove $y_c + \Delta y$ rappresenta la coordinata y di B: $y_B = y_c + \Delta y \rightarrow y_c = y_B - \Delta y$

e quindi:

$$\cos \alpha \cdot \frac{\Delta y}{\sqrt{y_c} + \sqrt{y_c - \Delta y}} = \cos \beta \cdot \frac{\Delta y}{\sqrt{y_B} + \sqrt{y_B - \Delta y}}.$$

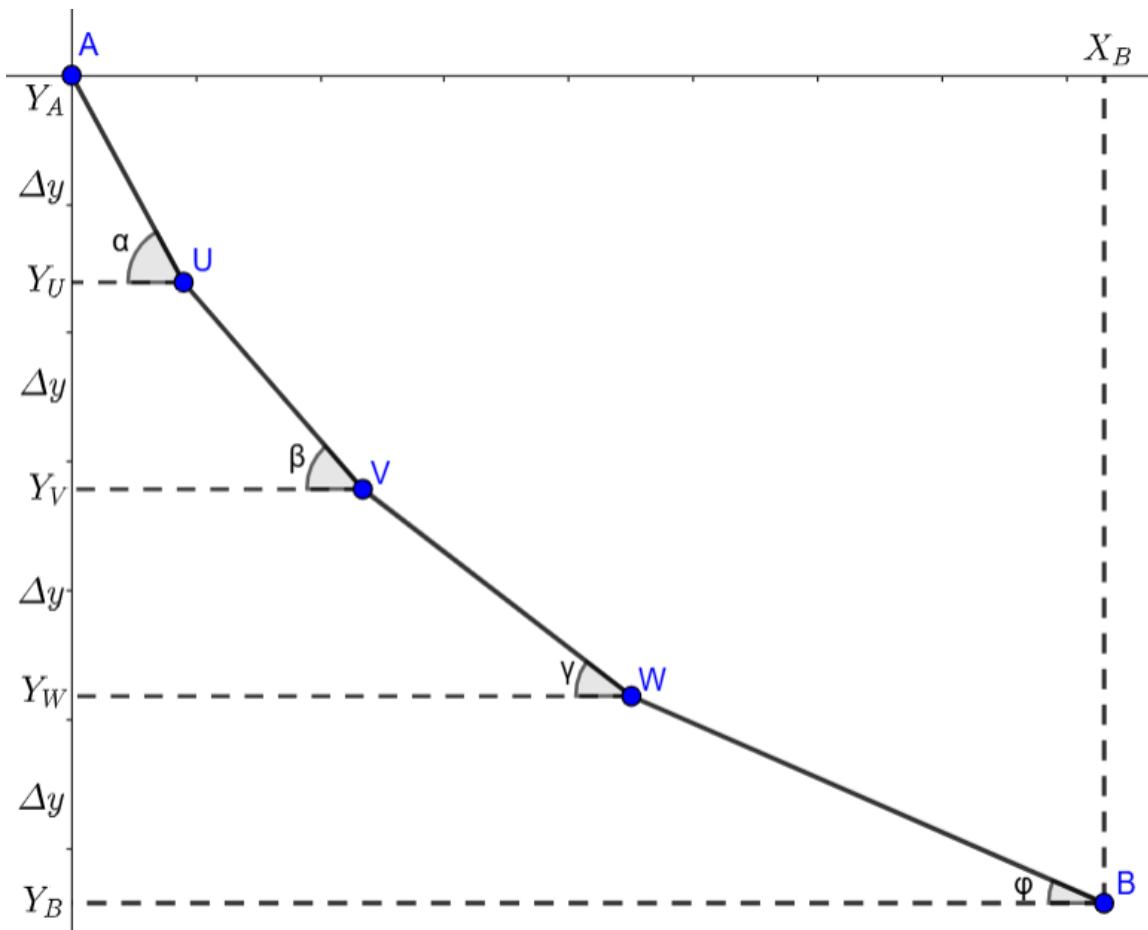
Δy è una quantità maggiore di 0 e pertanto si può semplificare

$$\frac{\cos \alpha}{\sqrt{y_c} + \sqrt{y_c - \Delta y}} = \frac{\cos \beta}{\sqrt{y_B} + \sqrt{y_B - \Delta y}}.$$

Si è ottenuta un'uguaglianza che stabilisce la costanza di un rapporto che coinvolge l'inclinazione del segmento e le quote y dei suoi estremi. Questa è una relazione generale che deve valere per qualsiasi spezzata di due segmenti a tempo minimo, indipendentemente dalla posizione del punto di arrivo rispetto al punto di partenza e dalla velocità iniziale di caduta. Tale relazione rappresenta una condizione analoga alla legge di Snell dell'ottica geometrica, che descrive la rifrazione di un raggio luminoso nella transizione tra due mezzi con indice di rifrazione differente. La legge di Snell discende dal principio di Fermat, secondo cui la luce segue il percorso che minimizza il tempo di percorrenza. Proprio questa analogia con la propagazione della luce ispirò Johann Bernoulli nella risoluzione del problema della brachistocrona. [1]

4.4. Generalizzazione a una spezzata con più segmenti

Si considera ora, invece che due soli segmenti, una spezzata costituita da n segmenti tali che, per ciascuno, la differenza tra la coordinata y del vertice superiore e quella del vertice inferiore è uguale a Δy . La spezzata che minimizza il tempo di caduta complessivo è quella costituita da tutte le coppie di segmenti che minimizzano il proprio tempo di caduta, ovvero la condizione precedente deve valere per ogni coppia di segmenti consecutivi. Si conclude quindi che il rapporto in questione è costante per tutti i segmenti della spezzata.

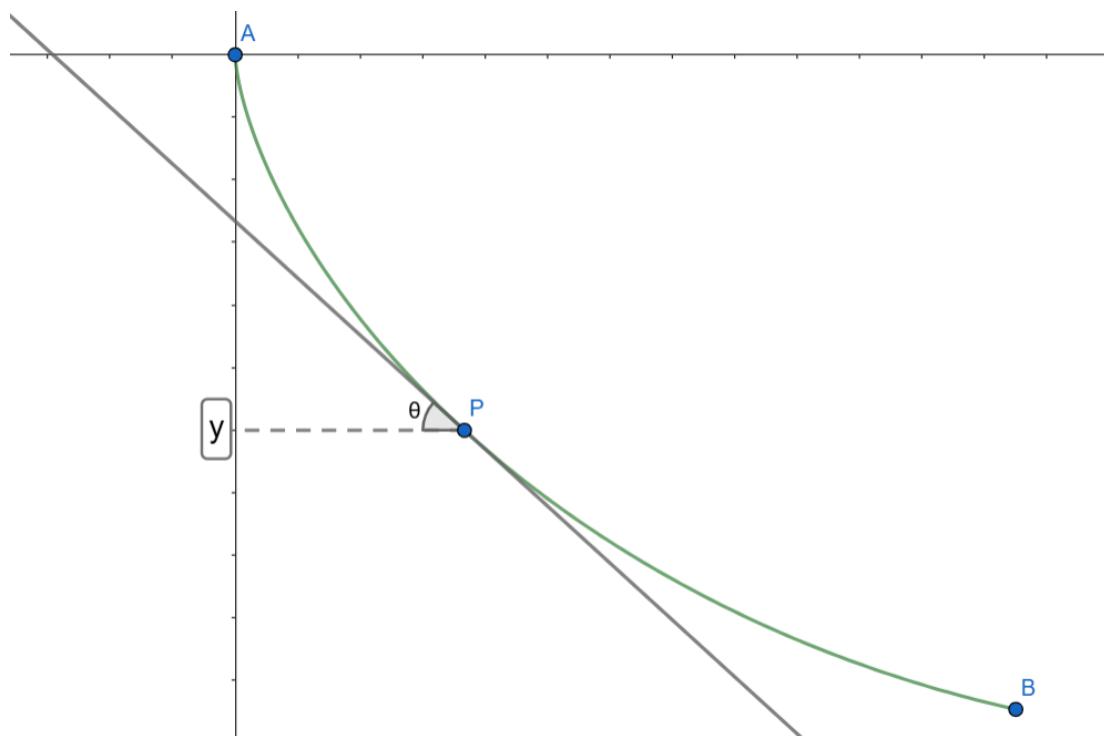


$$\frac{\cos \alpha}{(\sqrt{y_U} + \sqrt{y_U - \Delta y})} = \frac{\cos \beta}{(\sqrt{y_V} + \sqrt{y_V - \Delta y})} = \frac{\cos \gamma}{(\sqrt{y_W} + \sqrt{y_W - \Delta y})} = \frac{\cos \varphi}{(\sqrt{y_B} + \sqrt{y_B - \Delta y})}.$$

4.5. Condizione della brachistocrona

Al limite per un numero di segmenti $n \rightarrow \infty$ e quindi $\Delta y \rightarrow 0$, si ottiene che per ogni punto della brachistocrona vale $\frac{\cos \theta}{\sqrt{y}} = \text{costante}$, dove y rappresenta l'ordinata del punto generico e θ l'angolo acuto $\left(0 < \theta < \frac{\pi}{2}\right)$ tra la tangente alla brachistocrona nel punto e l'orizzontale. Indicando la costante con C ($C > 0$) si ha quindi $\frac{\cos \theta}{\sqrt{y}} = C$.

Questa viene chiamata la condizione della brachistocrona e deve valere in ogni suo punto.



Avvicinandosi ad A, $y \rightarrow 0$ e per mantenere il rapporto $\frac{\cos \theta}{\sqrt{y}} = C$, anche $\cos \theta \rightarrow 0$, da cui $\theta \rightarrow \frac{\pi}{2}$. Quindi in A, dove la massa parte da ferma, la curva brachistocrona ha tangente verticale, come ci si sarebbe aspettati intuitivamente affinché la massa sia sottoposta inizialmente alla massima accelerazione.

Si considera l'uguaglianza trigonometrica $\cos \theta = \frac{\cot \theta}{\sqrt{1 + (\cot \theta)^2}}$.

Per riuscire a determinare quale curva sia la brachistocrona, si cerca di ottenere dalla condizione, una funzione B per esprimere x in funzione di y : $x = B(y)$.

$\cot \theta$ si può considerare allora come la derivata prima di $B(y)$ rispetto a y ($\cot \theta = \frac{dx}{dy}$), da cui:

$$\cos \theta = \frac{\frac{dB(y)}{dy}}{\sqrt{1 + \left(\frac{dB(y)}{dy}\right)^2}}.$$

La funzione $B(y)$ non è detto che rappresenti la brachistocrona in tutto il suo dominio, però almeno localmente è la curva che soddisfa la condizione e pertanto minimizza il tempo di caduta nel suo tratto. Una volta ottenuta permette di identificare quale curva è la brachistocrona.

4.6. Risoluzione equazione differenziale

La condizione della brachistocrona si può esprimere così come un' equazione differenziale a variabili separabili:

$$\frac{\frac{dB(y)}{dy}}{\sqrt{1 + \left(\frac{dB(y)}{dy}\right)^2}} = C\sqrt{y}$$

$$\frac{\frac{dB(y)}{dy}}{\sqrt{1 + \left(\frac{dB(y)}{dy}\right)^2}} = C\sqrt{y}$$

$$\frac{\left(\frac{dB(y)}{dy}\right)^2}{1 + \left(\frac{dB(y)}{dy}\right)^2} = C^2 y$$

da cui si deduce che $0 < C^2 y < 1$,

$$1 + \frac{1}{\left(\frac{dB(y)}{dy}\right)^2} = \frac{1}{C^2 y}$$

$$\frac{1}{\left(\frac{dB(y)}{dy}\right)^2} = \frac{1}{C^2 y} - 1$$

$$\left(\frac{dB(y)}{dy}\right)^2 = \frac{C^2 y}{1 - C^2 y}.$$

Dall'osservazione iniziale la brachistocrona è una funzione continua di x . $B(y)$ è quindi invertibile. Essendo anche continua è dunque monotona, in particolare crescente. Quindi

$$\frac{dB(y)}{dy} > 0$$

$$\frac{dB(y)}{dy} = \frac{C\sqrt{y}}{\sqrt{1 - C^2 y}}$$

$$B(y) = \int \frac{C\sqrt{y}}{\sqrt{1 - C^2 y}} dy.$$

$$\text{Sostituendo } t = C\sqrt{y} \rightarrow y = \left(\frac{t}{C}\right)^2 \rightarrow dy = 2\frac{t}{C^2} dt.$$

Si ha $\begin{cases} 0 < t^2 < 1 \\ t > 0 \end{cases}$ quindi $0 < t < 1$. Inoltre, t è crescente perché y è crescente.

Si ottiene:

$$B(t) = \frac{2}{C^2} \int \frac{t^2}{\sqrt{1 - t^2}} dt$$

$$\text{e sostituendo } \sin z = t \rightarrow \cos z \cdot dz = dt.$$

$$t > 0 \rightarrow \sin z > 0 \rightarrow 0 < z < \pi. \quad t \text{ è crescente quindi } 0 < z < \frac{\pi}{2}.$$

$$B(z) = \frac{2}{C^2} \int \frac{(\sin z)^2}{\sqrt{1 - (\sin z)^2}} \cos z dz.$$

$$0 < z < \frac{\pi}{2} \rightarrow \cos z > 0. \quad \cos z \text{ si semplifica dunque con } \sqrt{1 - \sin z^2}.$$

Dalla formula trigonometrica $(\sin z)^2 = \frac{1-\cos 2z}{2}$:

$$B(z) = \frac{1}{C^2} \int (1 - \cos 2z) dz,$$

da cui si ottiene:

$$B(z) = \frac{1}{C^2} \cdot \left(z - \frac{\sin 2z}{2} \right) + D = \frac{1}{C^2} \cdot (z - \sin z \cdot \cos z) + D$$

con D costante di integrazione.

Risostituendo t :

$$B(t) = \frac{1}{C^2} \cdot \left(\sin^{-1} t - t \cdot \sqrt{1-t^2} \right) + D$$

e risostituendo y :

$$B(y) = \frac{1}{C^2} \cdot \left(\sin^{-1}(C\sqrt{y}) - C\sqrt{y} \cdot \sqrt{1-C^2y} \right) + D.$$

Imponendo il passaggio per l'origine si ha che $D = 0$.

Si vuole togliere la radice su y nell' argomento dell' arcoseno. Per fare ciò si considera una relazione esistente tra arcoseno e arcocoseno:

$$\begin{aligned} \text{ponendo } w &= \sin^{-1} x \rightarrow x = \sin w \rightarrow x^2 = (\sin w)^2 = \frac{1-\cos 2w}{2} \rightarrow \\ \cos 2w &= 1 - 2x^2 \rightarrow w = \frac{\cos^{-1}(1-2x^2)}{2} \rightarrow \sin^{-1} x = \frac{\cos^{-1}(1-2x^2)}{2}. \end{aligned}$$

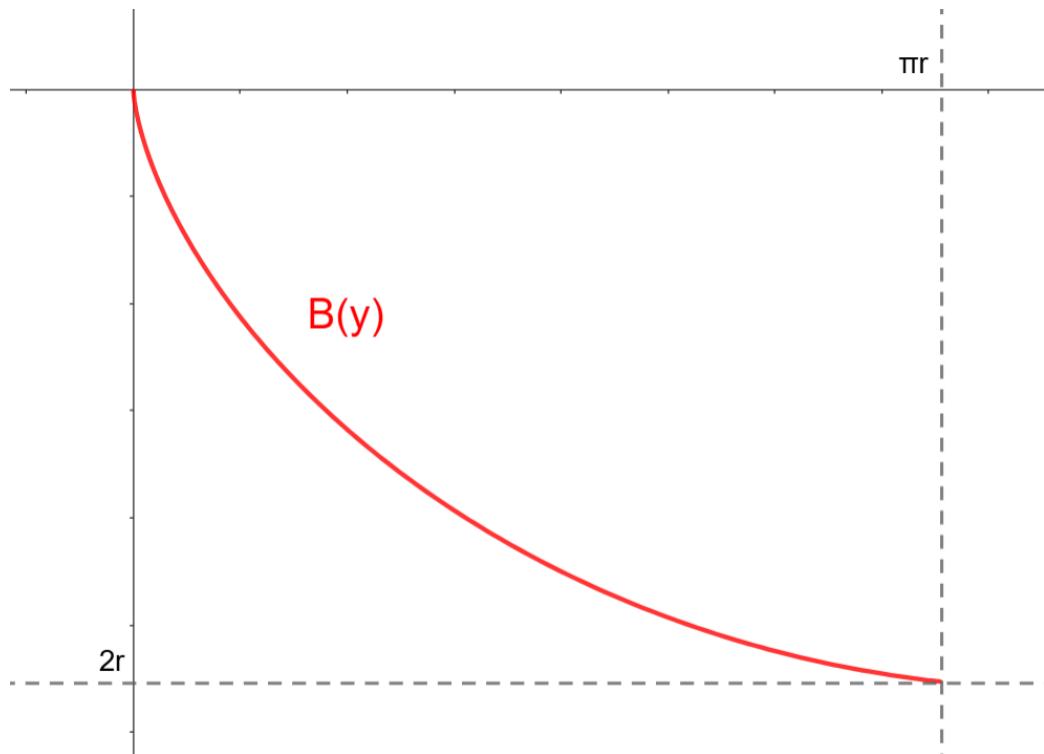
$B(y)$ diventa dunque:

$$B(y) = \frac{\cos^{-1}(1-2C^2y)}{2C^2} - \sqrt{y \cdot \left(\frac{1}{C^2} - y \right)}.$$

Se si pone $r = \frac{1}{2C^2}$ si ottiene:

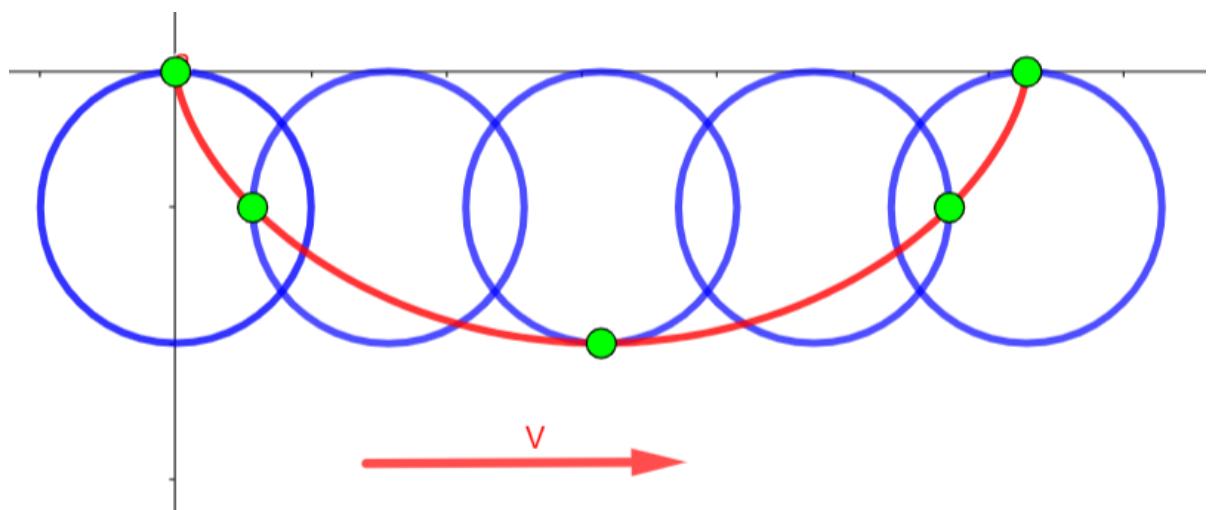
$$B(y) = r \cdot \cos^{-1} \left(1 - \frac{y}{r} \right) - \sqrt{y \cdot (2r-y)},$$

che si riconosce essere l'equazione cartesiana della cicloide.



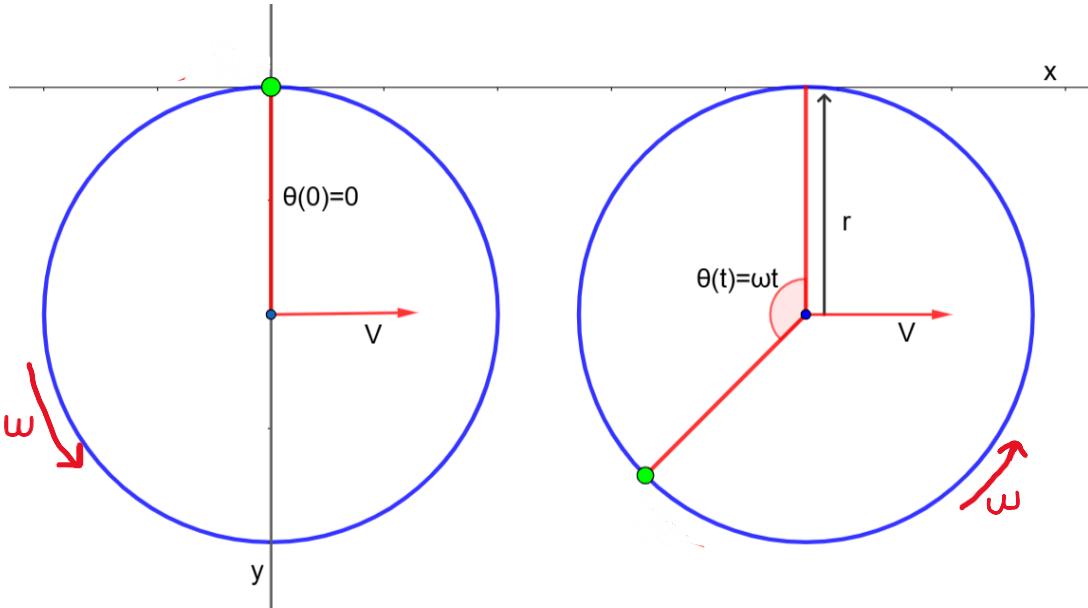
5. Cicloide

La cicloide è la curva tracciata da un punto fisso su una circonferenza che rotola, senza strisciare, lungo una retta.



Il moto del punto è la composizione di un moto circolare uniforme di velocità angolare ω e raggio della circonferenza r e di un moto rettilineo uniforme di velocità V nella direzione x .

Per il moto di rotolamento vale la seguente relazione tra velocità angolare e velocità lineare: $V = \omega r$.



5.1. Parametrizzazione

Si considera la cicloide passante per l'origine in $t = 0$. In forma parametrica $c(t)$ rispetto al tempo:

$$c(t) = \begin{cases} x = Vt - r \cdot \sin \omega t \\ y = r - r \cdot \cos \omega t \end{cases}$$

$$c(t) = \begin{cases} x = r \cdot (\omega t - \sin \omega t) \\ y = r \cdot (1 - \cos \omega t) \end{cases}.$$

$\varphi = \omega t$ rappresenta l'angolo di rotazione della circonferenza generatrice della cicloide.

$$c(\varphi) = \begin{cases} x = r \cdot (\varphi - \sin \varphi) \\ y = r \cdot (1 - \cos \varphi) \end{cases}.$$

5.2. Proprietà

Si considera la derivata prima rispetto a φ della componente x :

$$\frac{dx}{d\varphi} = r \cdot (1 - \cos \varphi) \geq 0 \quad \forall \varphi.$$

$x(\varphi)$ è quindi monotona crescente e, poiché continua, è invertibile. Ovvero, si può ottenere una funzione $\varphi(x)$, non necessariamente esprimibile con un'espressione esplicita. Si può dunque ottenere la funzione composta $y(\varphi(x))$. Da ciò si può concludere che la cicloide è una funzione di x .

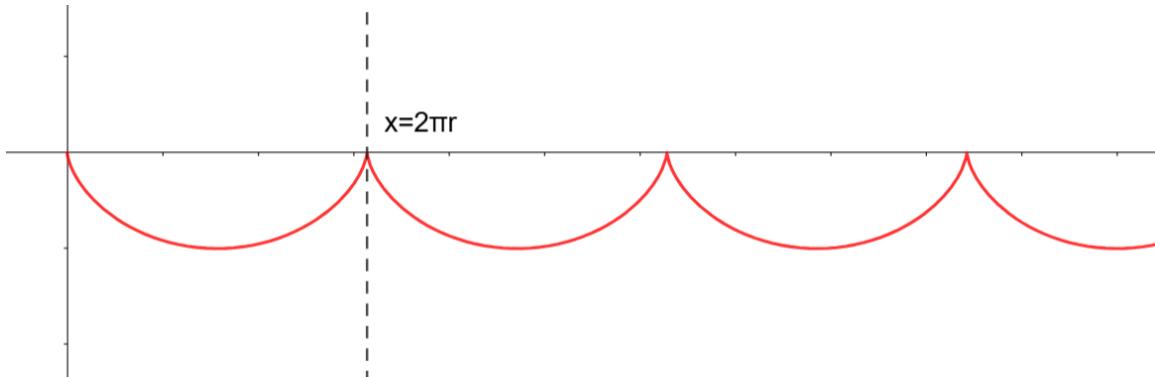
Inoltre, $y(\varphi)$ è periodica di periodo angolare 2π .

$$y(\varphi + 2\pi) = r \cdot (1 - \cos(\varphi + 2\pi)) = r \cdot (1 - \cos \varphi) = y(\varphi).$$

Un periodo temporale 2π corrisponde a un periodo spaziale per la componente x di $2\pi r$.

$$\begin{aligned} x(\varphi + 2\pi) &= r \cdot ((\varphi + 2\pi) - \sin(\varphi + 2\pi)) = r \cdot (\varphi + 2\pi - \sin(\varphi + 2\pi)) \\ &= r \cdot (\varphi - \sin \varphi) + 2\pi r = x(\varphi) + 2\pi r. \end{aligned}$$

Quindi $y(x)$ è una funzione periodica di periodo $2\pi r$.



Si studia la simmetria:

Si considera la riflessione del parametro φ rispetto al punto medio del periodo angolare π .

$$\begin{aligned} x(\pi - \varphi) &= r \cdot (\pi - \varphi - \sin(\pi - \varphi)) = r \cdot (\pi - \varphi - \sin \varphi) \\ &= r\pi - r \cdot (\varphi + \sin \varphi). \end{aligned}$$

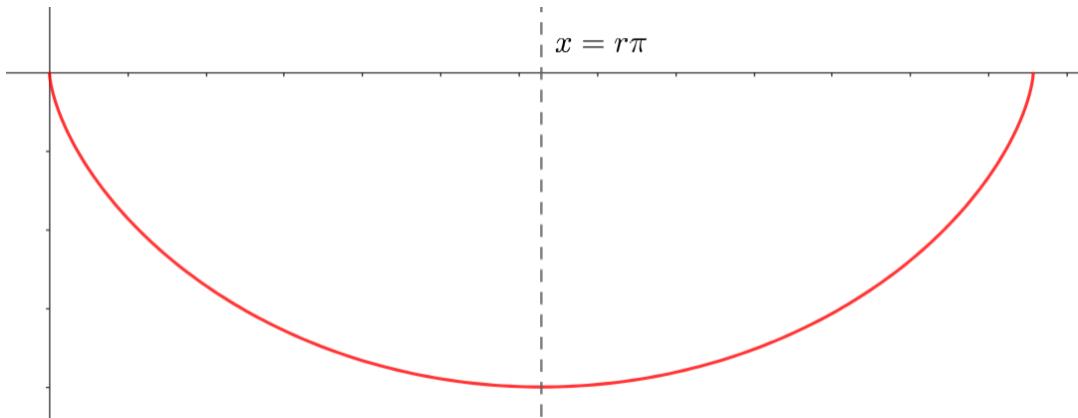
$$\begin{aligned}
x(\pi + \varphi) &= r \cdot (\pi + \varphi - \sin(\pi + \varphi)) = r \cdot (\pi + \varphi + \sin \varphi) \\
&= r\pi + r \cdot (\varphi + \sin \varphi).
\end{aligned}$$

Dunque, la simmetria rispetto al punto medio del periodo angolare corrisponde ad una simmetria rispetto al punto medio del periodo spaziale $r\pi$.

$$y(\pi - \varphi) = r \cdot (1 - \cos(\pi - \varphi)) = r \cdot (1 + \cos \varphi).$$

$$y(\pi + \varphi) = r \cdot (1 - \cos(\pi + \varphi)) = r \cdot (1 + \cos \varphi).$$

Si ha $y(\pi - \varphi) = y(\pi + \varphi)$ e quindi anche $y(r\pi - x) = y(r\pi + x)$: la cicloide è caratterizzata da una simmetria assiale di asse $x = r\pi$.



5.3. Equazione cartesiana

In $x = r \cdot (\varphi - \sin \varphi)$ non è possibile isolare φ rispetto a x . $y(x)$ è dunque una funzione implicita, cioè non è possibile esprimere y in forma esplicita come $y = f(x)$ utilizzando solo operazioni algebriche e funzioni elementari.

Limitando il dominio è possibile fare il contrario, esprimere cioè $x = f(y)$.

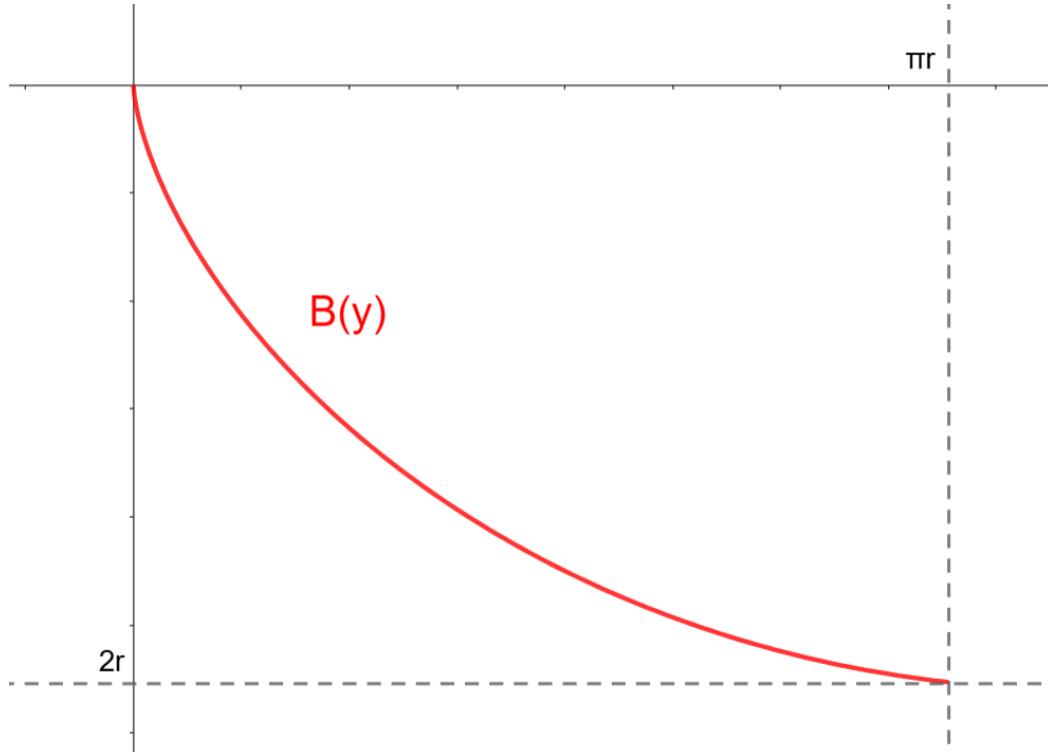
$$y = r \cdot (1 - \cos \varphi) \rightarrow \frac{y}{r} = 1 - \cos \varphi \rightarrow \cos \varphi = 1 - \frac{y}{r} \rightarrow \varphi = \cos^{-1}\left(1 - \frac{y}{r}\right).$$

Dato che il codominio di arcocoseno è $[0, \pi]$, il dominio angolare è limitato a $[0, \pi]$.

Si sostituisce ora nell'espressione di x :

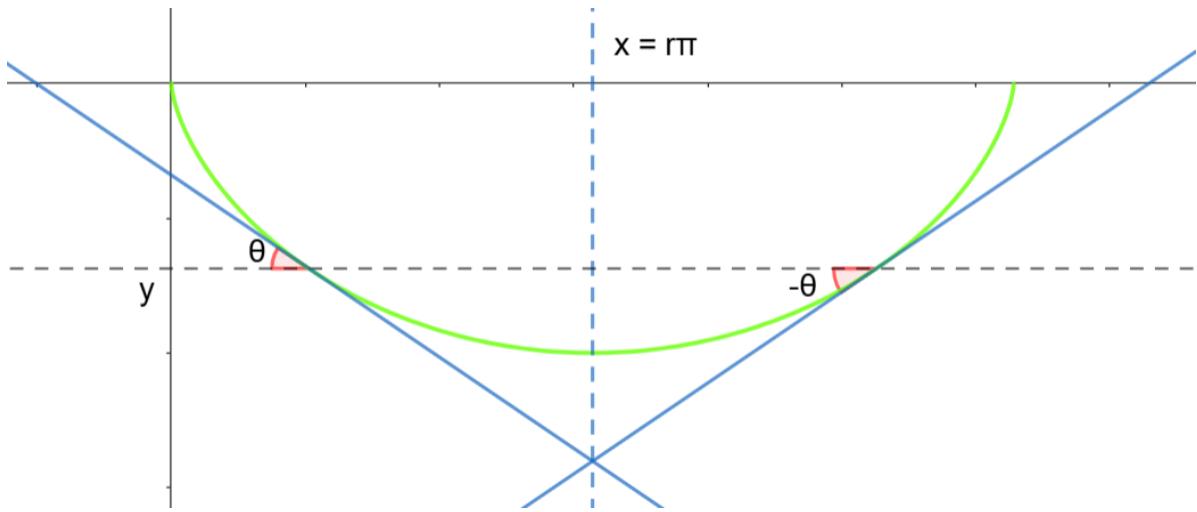
$$\begin{aligned}
 x &= r \cdot (t - \sin \varphi) = r \cdot \left(t - \sqrt{1 - (\cos \varphi)^2} \right) = \\
 &= r \cdot \left(\cos^{-1} \left(1 - \frac{y}{r} \right) - \sqrt{1 - \left(\cos \left(\cos^{-1} \left(1 - \frac{y}{r} \right) \right) \right)^2} \right) = \\
 &= r \cdot \left(\cos^{-1} \left(1 - \frac{y}{r} \right) - \sqrt{1 - \left(1 - \frac{y}{r} \right)^2} \right) = \\
 &= r \cdot \left(\cos^{-1} \left(1 - \frac{y}{r} \right) - \sqrt{\frac{r^2 - (r-y)^2}{r^2}} \right) = \\
 &= r \cdot \cos^{-1} \left(1 - \frac{y}{r} \right) - \sqrt{y \cdot (2r-y)} .
 \end{aligned}$$

Si è ottenuta così l'equazione cartesiana della cicloide, che corrisponde alla soluzione a cui si era giunti nella sezione precedente.



5.4. Verifica che la cicloide è la curva brachistocrona

Dato che il dominio in cui è definita l'equazione cartesiana $x(y)$ è metà periodo e che la cicloide è simmetrica rispetto all'asse verticale a metà del periodo, si può subito concludere che la condizione della brachistocrona $\frac{\cos \theta}{\sqrt{y}} = C$ è valida per tutto il periodo.



Si mostra che la condizione della brachistocrona $\frac{\cos \theta}{\sqrt{y}} = C$ si può ottenere dalla forma parametrica della cicloide, al fine di ottenere un' espressione per la quantità costante.

Si calcola la derivata:

$$c'(\varphi) = \begin{cases} x' = r \cdot (1 - \cos \varphi) \\ y' = r \cdot \sin \varphi \end{cases}.$$

Il coseno dell'angolo formato tra la tangente alla cicloide e l'orizzontale è:

$$\begin{aligned} \cos \theta &= \frac{x'}{\sqrt{x'^2 + y'^2}} = \frac{r \cdot (1 - \cos \varphi)}{\sqrt{(r \cdot (1 - \cos \varphi))^2 + (r \cdot \sin \varphi)^2}} = \\ &= \frac{(1 - \cos \varphi)}{\sqrt{1 - 2 \cos \varphi + (\cos \varphi)^2 + (\sin \varphi)^2}} = \frac{(1 - \cos \varphi)}{\sqrt{1 - 2 \cos \varphi + 1}} = \\ &= \frac{(1 - \cos \varphi)}{\sqrt{2 \cdot (1 - \cos \varphi)}} = \frac{1}{\sqrt{2}} \cdot \sqrt{1 - \cos \varphi}. \end{aligned}$$

La condizione della brachistocrona:

$$\frac{\cos \theta}{\sqrt{y}} = \frac{1}{\sqrt{2}} \cdot \frac{\sqrt{1 - \cos \varphi}}{\sqrt{r \cdot (1 - \cos \varphi)}} = \frac{1}{\sqrt{2r}}.$$

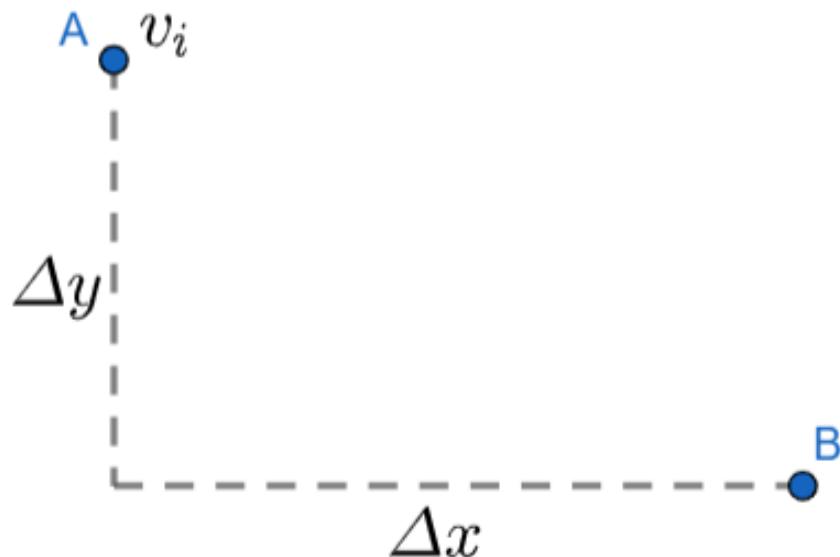
Si è ottenuto che per ogni punto il rapporto tra il coseno dell'angolo tra la tangente alla cicloide in quel punto e l'orizzontale, e la radice quadrata della coordinata y del punto è costante; e tale costante dipende dal raggio della circonferenza generatrice della cicloide.

5.5. Determinazione del ramo di cicloide che rappresenta la curva brachistocrona

Si conclude considerando le condizioni di passaggio della cicloide per i punti di partenza e arrivo del problema della brachistocrona, in modo che la curva brachistocrona per ogni configurazione dei punti di partenza e arrivo sia univocamente determinata.

Una particolare configurazione del problema della brachistocrona è definita dalle distanze tra punto di partenza A e di arrivo B lungo x e lungo y che si indicano rispettivamente con Δx e Δy . Generalizzando il problema si può considerare la situazione con velocità iniziale non nulla v_i nel punto di partenza.

I dati noti del problema sono dunque: Δx , Δy e v_i .



Si sceglie il sistema di coordinate in modo comodo, per avere la seguente relazione per la velocità: $v(y) = \sqrt{2gy}$, ovvero, se si considera un grave che parte da fermo ($v = 0$) nel punto di partenza, questo avrà coordinata $Y_A = 0$.

Invece se si considera una velocità iniziale v_i diversa da 0, il punto di partenza avrà coordinata $Y_A = \frac{v_i^2}{2g}$.

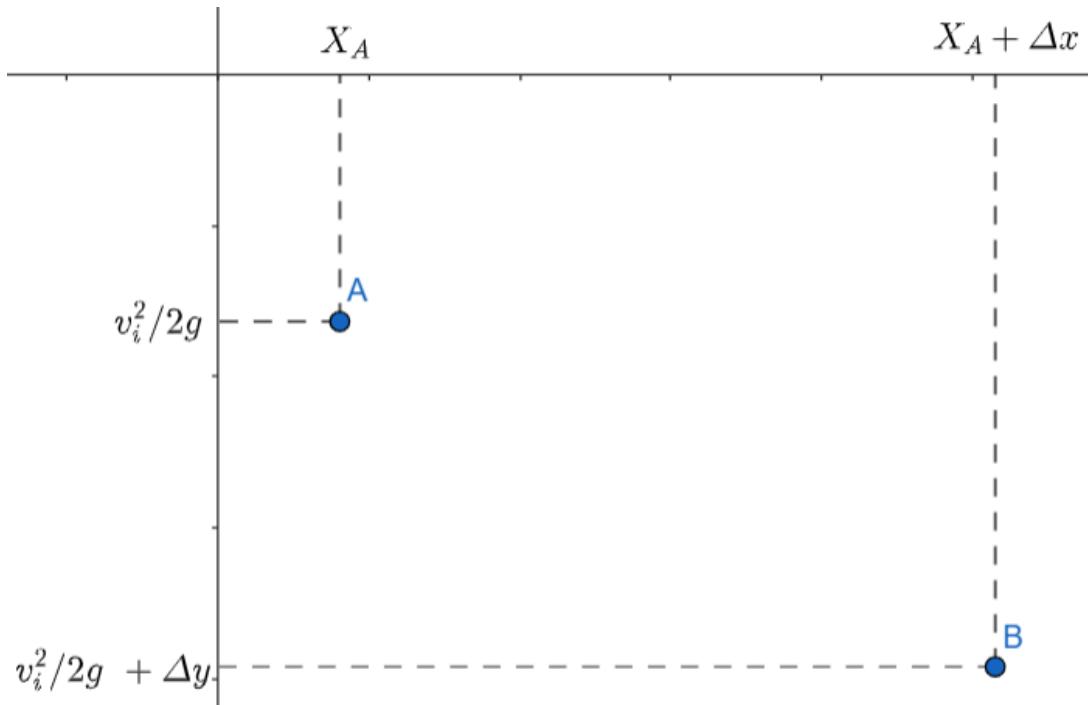
Le distanze tra punto di partenza e arrivo lungo x e lungo y si indicano rispettivamente con $\Delta x = X_B - X_A$ e $\Delta y = Y_B - Y_A$.

X_A, X_B, Y_A, Y_B si considera abbiano tutti valori positivi, mentre $\Delta x, \Delta y$ possono assumere anche valori negativi.

Δx può assumere qualsiasi valore sia positivo che negativo.

Δy può assumere qualsiasi valore positivo mentre se è negativo deve essere $\Delta y \geq -Y_A$, poiché $Y_A \geq 0$.

Δy può anche essere negativo e cioè il punto di arrivo B può essere più in alto del punto di partenza A, perché si sta considerando il caso generale in cui nel punto di partenza si ha una velocità iniziale.



È dunque necessario ricavare due incognite:

- l'ascissa X_A del punto di partenza per poter individuare l'origine del sistema di riferimento.
- Il raggio r della cicloide.

Si impongono i passaggi per i valori noti delle ordinate, ottenendo così i valori φ_A e φ_B a cui corrispondono i punti A e B, in funzione di r .

$$Y_A = r \cdot (1 - \cos \varphi_A) \rightarrow \cos \varphi_A = 1 - \frac{Y_A}{r}.$$

$$\varphi_A = \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \text{ o } \varphi_A = 2\pi - \cos^{-1} \left(1 - \frac{Y_A}{r} \right).$$

Allo stesso modo:

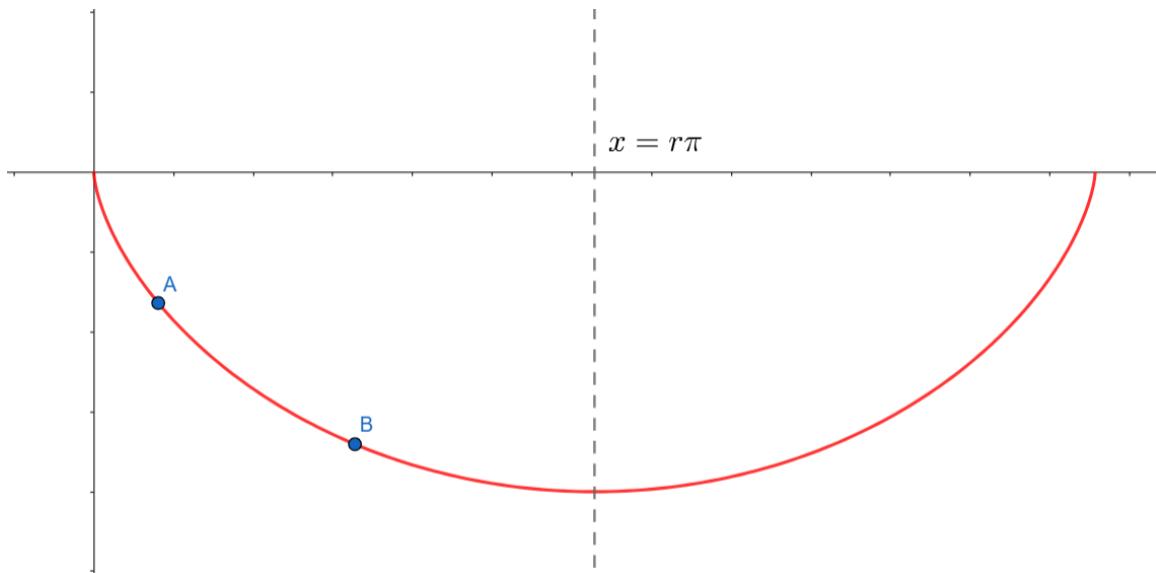
$$\varphi_B = \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \text{ o } \varphi_B = 2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right).$$

Si hanno dunque quattro possibilità a seconda del semiperiodo in cui si trovano A e B:

$$1) \quad \varphi_A = \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \text{ e } \varphi_B = \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right)$$

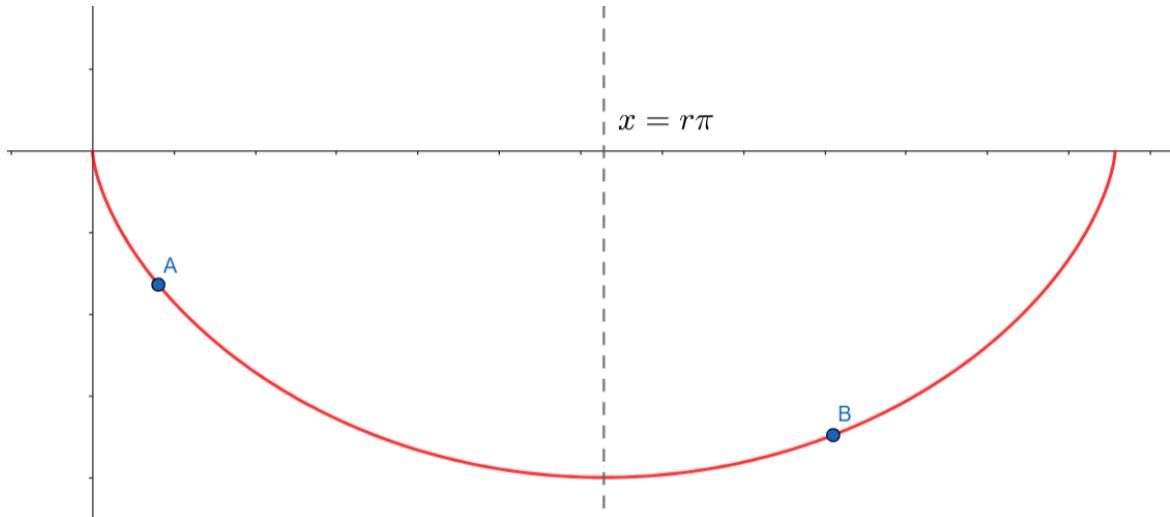
entrambi A e B nel semiperiodo $[0, r\pi]$.

Poiché nel semiperiodo $[0, r\pi]$ la cicloide è crescente, questo caso vale per Δx e Δy entrambi > 0 oppure per Δx e Δy entrambi < 0 (Δx e Δy concordi).



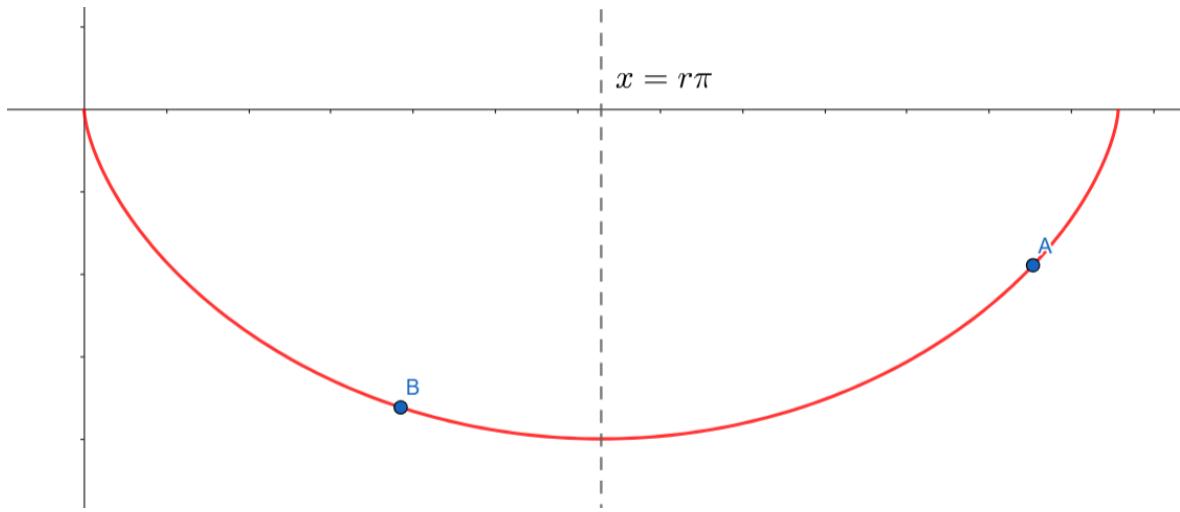
$$2) \quad \varphi_A = \cos^{-1} \left(1 - \frac{y_A}{r} \right) \quad \text{e} \quad \varphi_B = 2\pi - \cos^{-1} \left(1 - \frac{y_A + \Delta y}{r} \right).$$

A nel semiperiodo $[0, r\pi]$, B nel semiperiodo $[r\pi, 2r\pi]$, dunque questo caso vale per $\Delta x > 0$.



$$3) \quad \varphi_A = 2\pi - \cos^{-1} \left(1 - \frac{y_A}{r} \right) \quad \text{e} \quad \varphi_B = \cos^{-1} \left(1 - \frac{y_A + \Delta y}{r} \right).$$

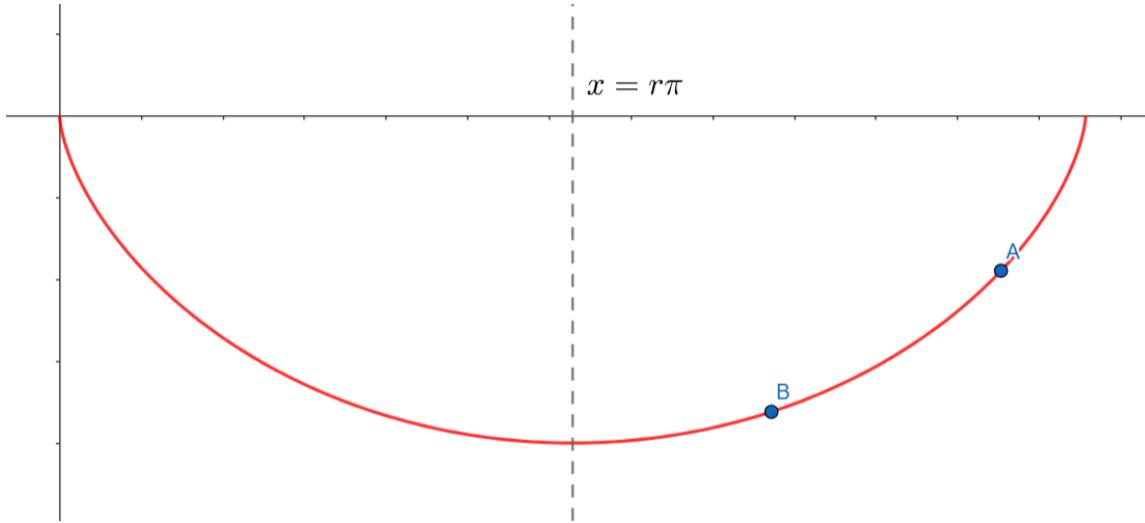
A nel semiperiodo $[r\pi, 2r\pi]$, B nel semiperiodo $[0, r\pi]$, dunque questo caso vale per $\Delta x < 0$.



$$4) \quad \varphi_A = 2\pi - \cos^{-1} \left(1 - \frac{y_A}{r} \right) \quad \text{e} \quad \varphi_B = 2\pi - \cos^{-1} \left(1 - \frac{y_A + \Delta y}{r} \right)$$

entrambi A e B nel semiperiodo $[r\pi, 2r\pi]$.

Poiché nel semiperiodo $[r\pi, 2r\pi]$ la cicloide è decrescente, questo caso vale per Δx e Δy discordi.



Ricapitolando si possono selezionare a priori alcune delle quattro alternative secondo i seguenti criteri:

	$\Delta x > 0$	$\Delta x < 0$	$\Delta x = 0$
$\Delta y > 0$	1) o 2)	3) o 4)	Cicloide degener: ($r = \infty$) segmento verticale
$\Delta y < 0$	2) o 4)	1) o 3)	Cicloide degener: ($r = \infty$) segmento verticale
$\Delta y = 0$	2)	3)	Punti coincidenti: r indeterminato

Si impone ora la condizione su Δx :

$$\Delta x = X_B - X_A = r \cdot (\varphi_B - \sin \varphi_B) - r \cdot (\varphi_A - \sin \varphi_A) = r \cdot (\varphi_B - \varphi_A - (\sin \varphi_B - \sin \varphi_A)).$$

Si valutano tutte le quattro possibilità di φ_A e φ_B :

$$1) \quad \Delta x = r \cdot \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) - \left(\sin \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \sin \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) \right) \right).$$

$$\text{Ora: } \sin \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) = \sqrt{1 - \left(\cos \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) \right)^2} = \sqrt{1 - \left(1 - \frac{Y_A}{r} \right)^2} = \\ = \sqrt{2 \frac{Y_A}{r} - \left(\frac{Y_A}{r} \right)^2} = \frac{\sqrt{2 Y_A r - Y_A^2}}{r} = \frac{\sqrt{Y_A \cdot (2r - Y_A)}}{r}.$$

Da cui:

$$\Delta x = r \cdot \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) - \sqrt{(Y_A + \Delta y) \cdot (2r - (Y_A + \Delta y))} + \sqrt{Y_A \cdot (2r - Y_A)} \right).$$

$$2) \quad \Delta x = r \cdot \left(\left(2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) - \left(\sin \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \sin \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) \right) \right).$$

$$\Delta x = r \cdot \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) - \left(-\sin \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \sin \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) \right) \right).$$

$$\Delta x = r \cdot \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) + \sqrt{(Y_A + \Delta y) \cdot (2r - (Y_A + \Delta y))} + \sqrt{Y_A \cdot (2r - Y_A)} \right).$$

$$3) \quad \Delta x = r \cdot \left(\left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) - \left(\sin \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \sin \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) \right) \right).$$

$$\Delta x = r \cdot \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) - 2\pi + \cos^{-1} \left(1 - \frac{Y_A}{r} \right) - \left(\sin \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) + \sin \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) \right) \right).$$

$$\Delta x = r \cdot \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) + \cos^{-1} \left(1 - \frac{Y_A}{r} \right) - 2\pi \right) - \sqrt{(Y_A + \Delta y) \cdot (2r - (Y_A + \Delta y))} - \sqrt{Y_A \cdot (2r - Y_A)}.$$

$$4) \quad \Delta x = r \cdot \left(\left(2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) - \left(\sin \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) - \sin \left(2\pi - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) \right) \right).$$

$$\Delta x = r \cdot \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) - \left(\sin \left(\cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) - \sin \left(\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) \right) \right) \right).$$

$$\Delta x = r \cdot \left(-\cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) + \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \right) + \sqrt{(Y_A + \Delta y) \cdot (2r - (Y_A + \Delta y))} - \sqrt{Y_A \cdot (2r - Y_A)}.$$

Si ottengono così quattro equazioni della forma: $\Delta x = f_i(r)$.

Si nota che $f_3(r) = -f_2(r)$ e $f_4(r) = -f_1(r)$.

Sono equazioni nell'incognita r parametriche rispetto a Δx , Δy e Y_A .

Queste equazioni non possono essere risolte analiticamente poiché l'incognita compare sia nella radice quadrata che nell'arcocoseno e, pertanto, non è possibile isolare la r . Devono quindi essere risolte numericamente.

Sulla base di Δx e Δy , secondo i criteri, si selezionano al più due equazioni. Una delle due risulterà impossibile, mentre l'altra ammetterà come unica soluzione il raggio della cicloide cercato.

Una volta ottenuto il raggio r è possibile ottenere anche $X_A = r \cdot (\varphi_A - \sin \varphi_A)$,

con φ_A corrispondente all'equazione da cui si ha ottenuto il raggio.

$$\varphi_A = \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \text{ per 1) e 2).}$$

$$\varphi_A = 2\pi - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) \text{ per 3) e 4).}$$

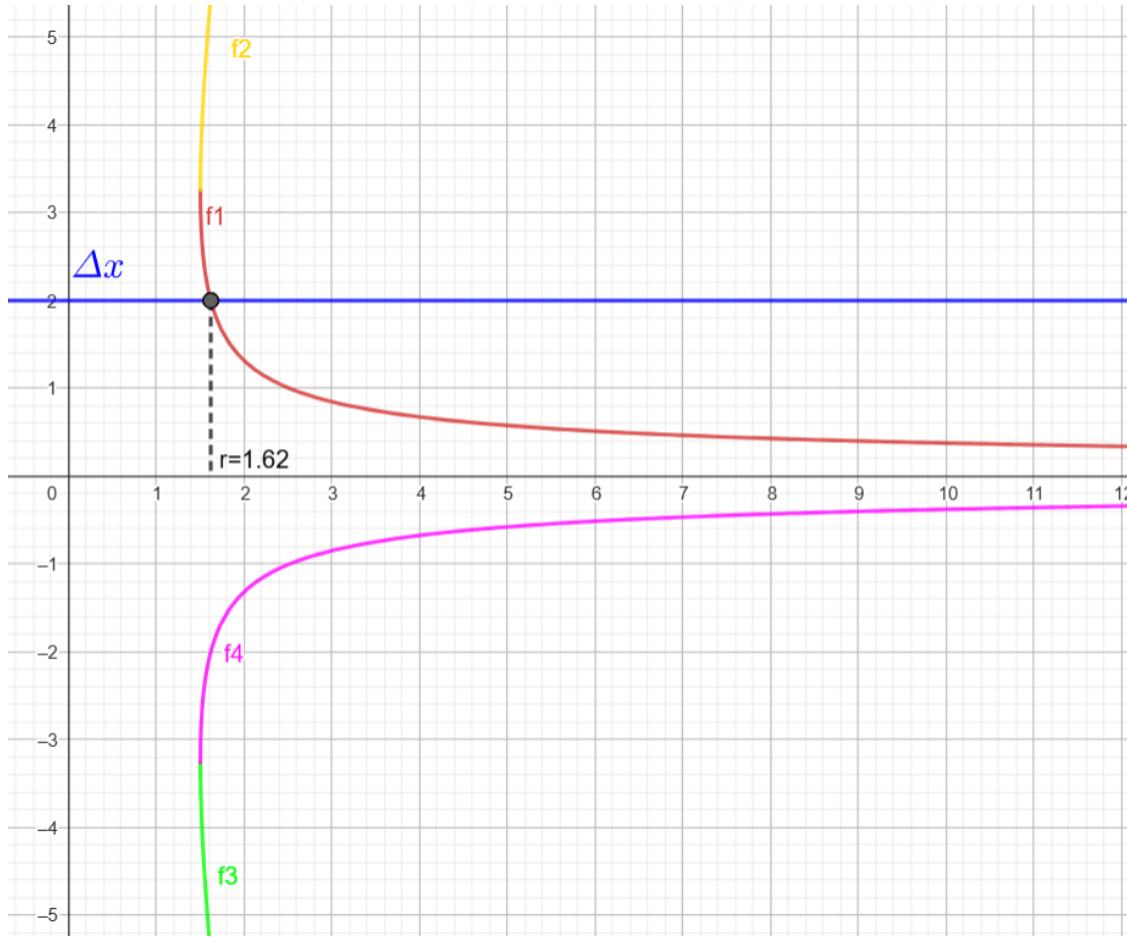
Esempi:

$$1) \quad Y_A = 2, \quad \Delta x = 2, \quad \Delta y = 1.$$

Dai criteri di selezione delle equazioni si considerano a priori solo f_1 e f_2 .

$$f_1 = r \cdot \left(\cos^{-1} \left(1 - \frac{3}{r} \right) - \cos^{-1} \left(1 - \frac{2}{r} \right) \right) - \sqrt{3 \cdot (2r - 3)} + \sqrt{2 \cdot (2r - 2)}.$$

$$f_2 = r \cdot \left(2\pi - \cos^{-1} \left(1 - \frac{3}{r} \right) - \cos^{-1} \left(1 - \frac{2}{r} \right) \right) + \sqrt{3 \cdot (2r - 3)} + \sqrt{2 \cdot (2r - 2)}.$$



Per f_1 la soluzione è $r \cong 1.62$, mentre per f_2 l'equazione risulta essere impossibile.

Si valutano ora i parametri φ_A e φ_B :

$$\varphi_A = \cos^{-1} \left(1 - \frac{Y_A}{r} \right) = \cos^{-1} \left(1 - \frac{2}{1.62} \right) \cong 1.81.$$

$$\varphi_B = \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) = \cos^{-1} \left(1 - \frac{2+1}{1.62} \right) \cong 2.59.$$

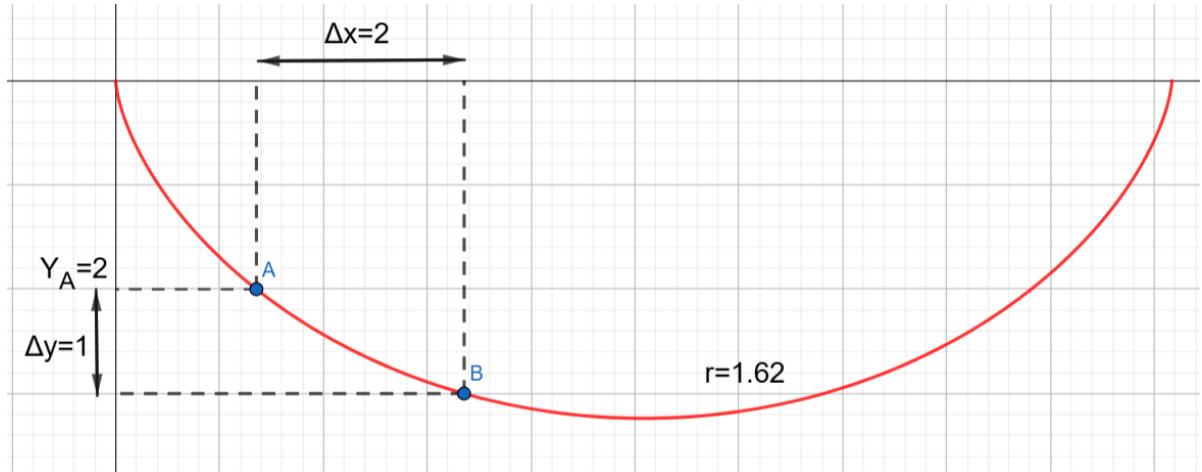
Si calcolano le coordinate di A e B :

$$X_A = r \cdot (\varphi_A - \sin \varphi_A) = 1.62 \cdot (1.81 - \sin 1.81) \cong 1.36 .$$

$$Y_A = r \cdot (1 - \cos \varphi_A) = 1.62 \cdot (1 - \cos 1.81) \cong 2 = Y_A .$$

$$X_B = r \cdot (\varphi_B - \sin \varphi_B) = 1.62 \cdot (2.59 - \sin 2.59) \cong 3.36 = X_{A1} + \Delta x .$$

$$Y_B = r \cdot (1 - \cos \varphi_A) = 1.62 \cdot (1 - \cos 2.59) \cong 3 = Y_A + \Delta y .$$

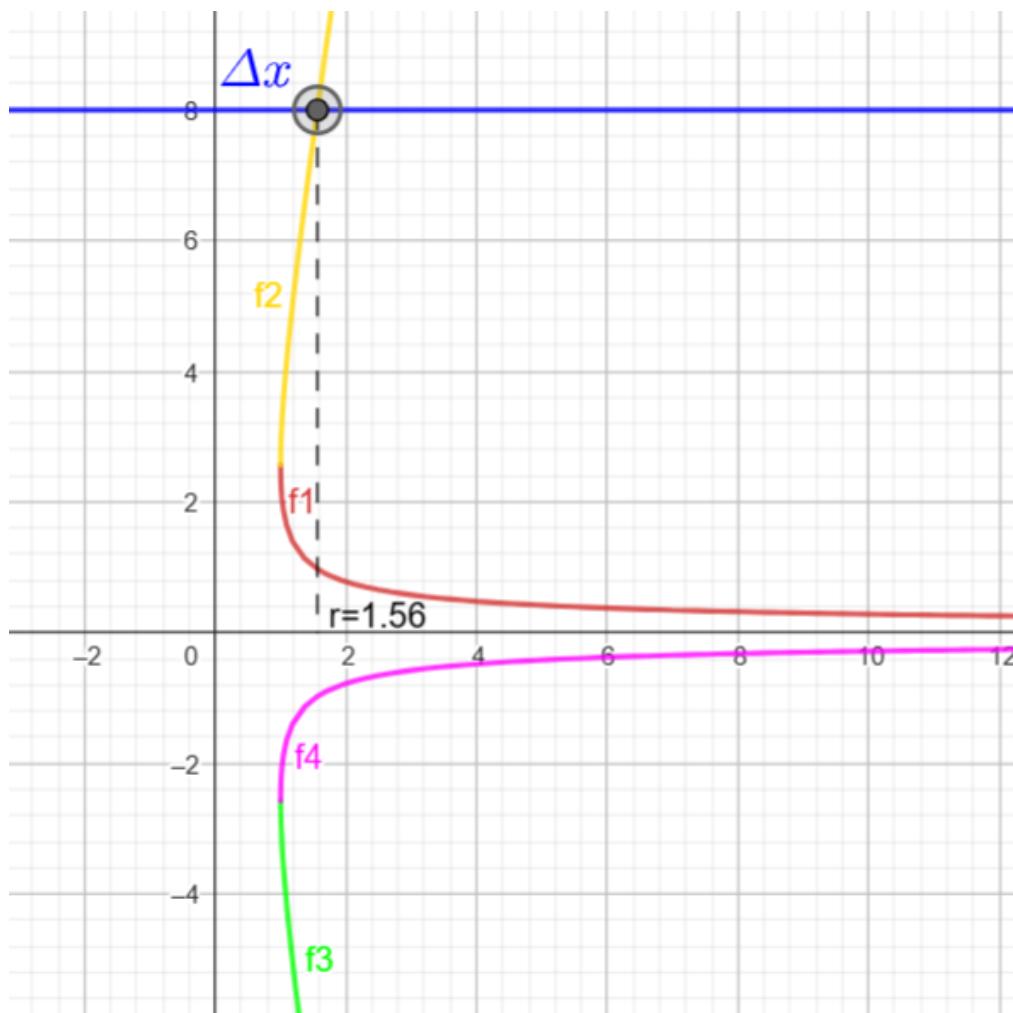


$$2) \quad Y_A = 1, \quad \Delta x = 8, \quad \Delta y = 1 .$$

Dai criteri di selezione delle equazioni si considerano a priori solo f_1 e f_2 .

$$f_1 = r \cdot \left(\cos^{-1} \left(1 - \frac{2}{r} \right) - \cos^{-1} \left(1 - \frac{1}{r} \right) \right) - \sqrt{2 \cdot (2r - 2)} + \sqrt{(2r - 1)} .$$

$$f_2 = r \cdot \left(2\pi - \cos^{-1} \left(1 - \frac{2}{r} \right) - \cos^{-1} \left(1 - \frac{1}{r} \right) \right) + \sqrt{2 \cdot (2r - 2)} + \sqrt{3 \cdot (2r - 1)} .$$



Per f_2 la soluzione è $r \cong 1.56$, mentre per f_1 l'equazione risulta essere impossibile.

Si valutano ora i parametri φ_A e φ_B :

$$\varphi_A = \cos^{-1}\left(1 - \frac{y_A}{r}\right) = \cos^{-1}\left(1 - \frac{1}{1.56}\right) \cong 1.20 .$$

$$\varphi_B = 2\pi - \cos^{-1}\left(1 - \frac{y_A + \Delta y}{r}\right) = 2\pi - \cos^{-1}\left(1 - \frac{1+1}{1.56}\right) \cong 4.43 .$$

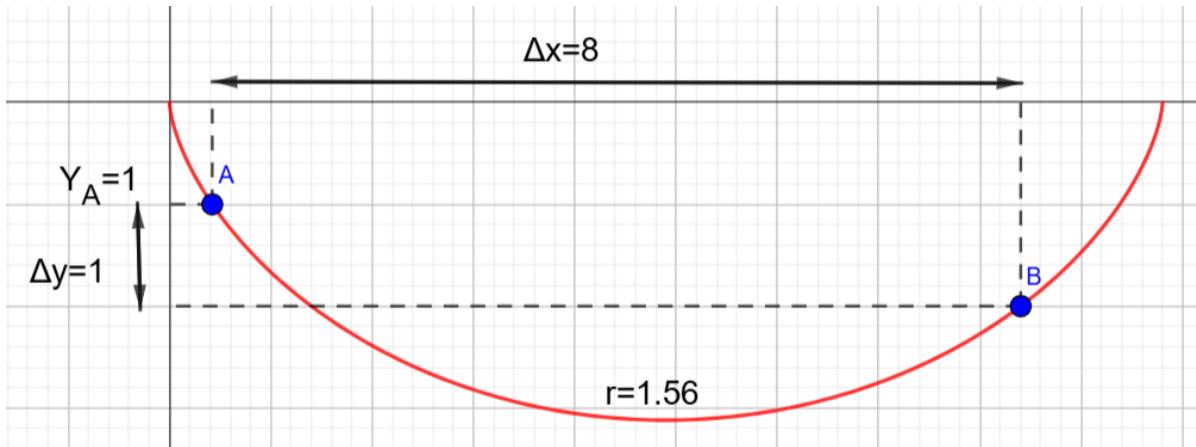
Si calcolano le coordinate di A e B :

$$X_A = r \cdot (\varphi_A - \sin \varphi_A) = 1.56 \cdot (1.20 - \sin 1.20) \cong 0.42 .$$

$$Y_A = r \cdot (1 - \cos \varphi_A) = 1.56 \cdot (1 - \cos 1.20) \cong 1 = Y_A .$$

$$X_B = r \cdot (\varphi_B - \sin \varphi_B) = 1.56 \cdot (4.43 - \sin 4.43) \cong 8.42 = X_{A1} + \Delta x .$$

$$Y_B = r \cdot (1 - \cos \varphi_B) = 1.56 \cdot (1 - \cos 4.43) \cong 2 = Y_A + \Delta y .$$

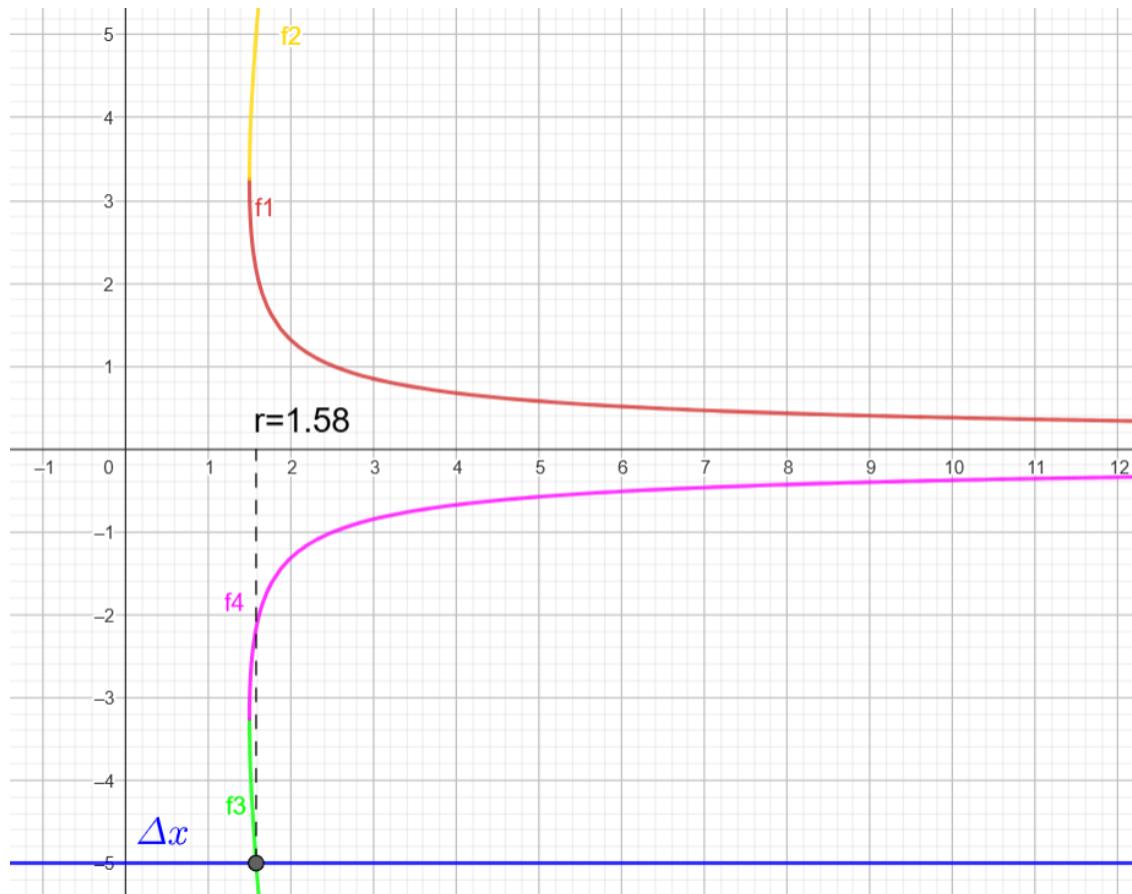


$$3) Y_A = 2, \Delta x = -5, \Delta y = 1.$$

Dai criteri di selezione delle equazioni si considerano a priori solo f_3 e f_4 .

$$f_3 = r \cdot \left(\cos^{-1} \left(1 - \frac{3}{r} \right) + \cos^{-1} \left(1 - \frac{2}{r} \right) - 2\pi \right) - \sqrt{3 \cdot (2r - 3)} - \sqrt{(2r - 2)} .$$

$$f_4 = r \cdot \left(-\cos^{-1} \left(1 - \frac{3}{r} \right) + \cos^{-1} \left(1 - \frac{2}{r} \right) \right) + \sqrt{3 \cdot (2r - 3)} - \sqrt{(2r - 2)} .$$



Per f_3 la soluzione è $r \cong 1.58$, mentre per f_4 l'equazione risulta essere impossibile.

Si valutano i parametri φ_A e φ_B :

$$\varphi_A = 2\pi - \cos^{-1}\left(1 - \frac{Y_A}{r}\right) = 2\pi - \cos^{-1}\left(1 - \frac{2}{1.58}\right) \cong 4.44 .$$

$$\varphi_B = \cos^{-1}\left(1 - \frac{Y_A + \Delta y}{r}\right) = \cos^{-1}\left(1 - \frac{2+1}{1.58}\right) \cong 2.69 .$$

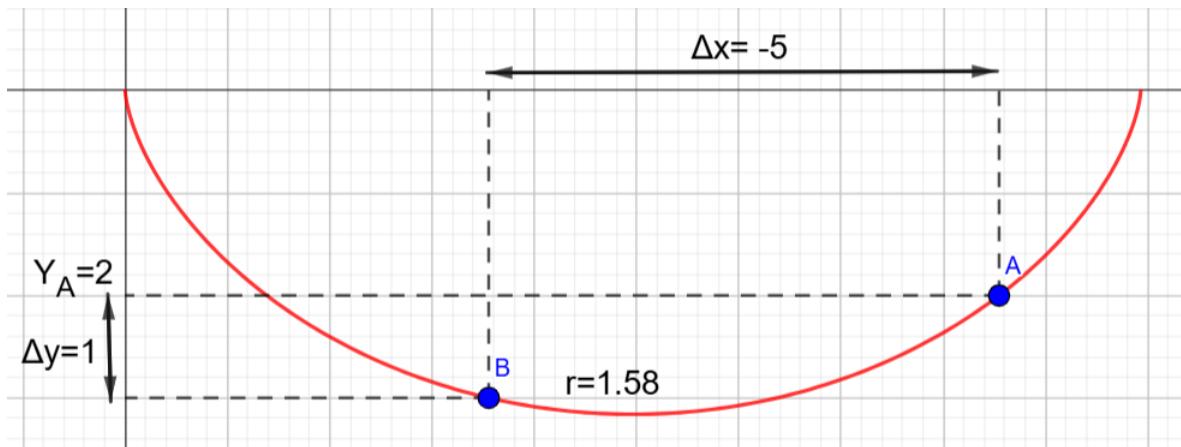
Si calcolano le coordinate di A e B :

$$X_A = r \cdot (\varphi_A - \sin \varphi_A) = 1.58 \cdot (4.44 - \sin 4.44) \cong 8.54 .$$

$$Y_A = r \cdot (1 - \cos \varphi_A) = 1.58 \cdot (1 - \cos 4.44) \cong 2 = Y_A .$$

$$X_B = r \cdot (\varphi_B - \sin \varphi_B) = 1.58 \cdot (2.69 - \sin 2.69) \cong 3.54 = X_{A1} + \Delta x .$$

$$Y_B = r \cdot (1 - \cos \varphi_B) = 1.58 \cdot (1 - \cos 2.69) \cong 3 = Y_A + \Delta y .$$

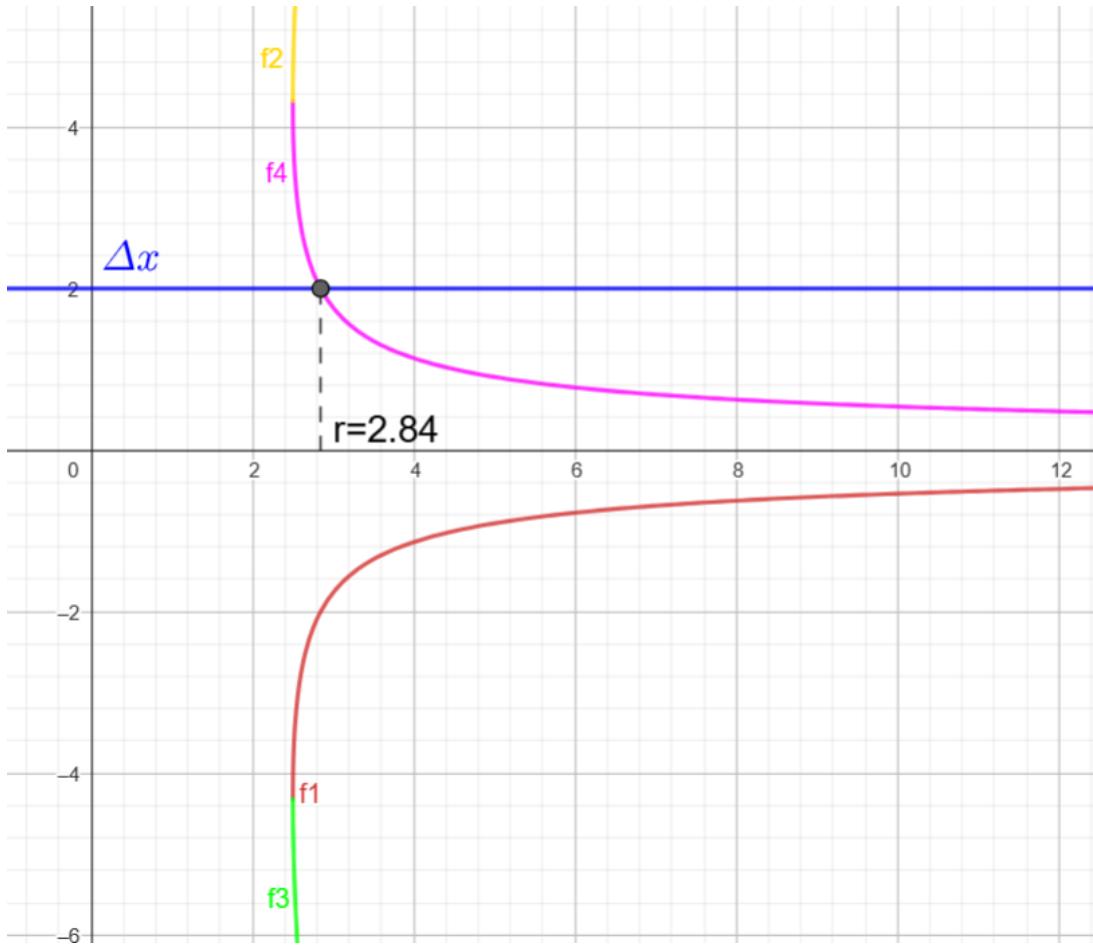


$$4) \quad Y_A = 5, \quad \Delta x = 2, \quad \Delta y = -1 .$$

Dai criteri di selezione delle equazioni si considerano a priori solo f_2 e f_4 .

$$f_2 = r \cdot \left(2\pi - \cos^{-1}\left(1 - \frac{4}{r}\right) - \cos^{-1}\left(1 - \frac{5}{r}\right)\right) + \sqrt{4 \cdot (2r - 4)} + \sqrt{5 \cdot (2r - 5)} .$$

$$f_4 = r \cdot \left(-\cos^{-1}\left(1 - \frac{4}{r}\right) + \cos^{-1}\left(1 - \frac{5}{r}\right)\right) + \sqrt{4 \cdot (2r - 4)} - \sqrt{5 \cdot (2r - 5)} .$$



Per f_4 la soluzione è $r \cong 2.84$, mentre per f_2 l'equazione risulta essere impossibile.

Si valutano i parametri φ_A e φ_B :

$$\varphi_A = 2\pi - \cos^{-1} \left(1 - \frac{Y_A}{r} \right) = 2\pi - \cos^{-1} \left(1 - \frac{5}{2.84} \right) \cong 3.85.$$

$$\varphi_B = 2\pi - \cos^{-1} \left(1 - \frac{Y_A + \Delta y}{r} \right) = 2\pi - \cos^{-1} \left(1 - \frac{5-1}{2.84} \right) \cong 4.29.$$

Si calcolano le coordinate di A e B :

$$X_A = r \cdot (\varphi_A - \sin \varphi_A) = 2.84 \cdot (3.85 - \sin 3.85) \cong 12.78.$$

$$Y_A = r \cdot (1 - \cos \varphi_A) = 2.84 \cdot (1 - \cos 3.85) \cong 5 = Y_A.$$

$$X_B = r \cdot (\varphi_B - \sin \varphi_B) = 2.84 \cdot (4.29 - \sin 4.29) \cong 14.78 = X_{A1} + \Delta x.$$

$$Y_B = r \cdot (1 - \cos \varphi_B) = 2.84 \cdot (1 - \cos 4.29) \cong 4 = Y_A + \Delta y.$$

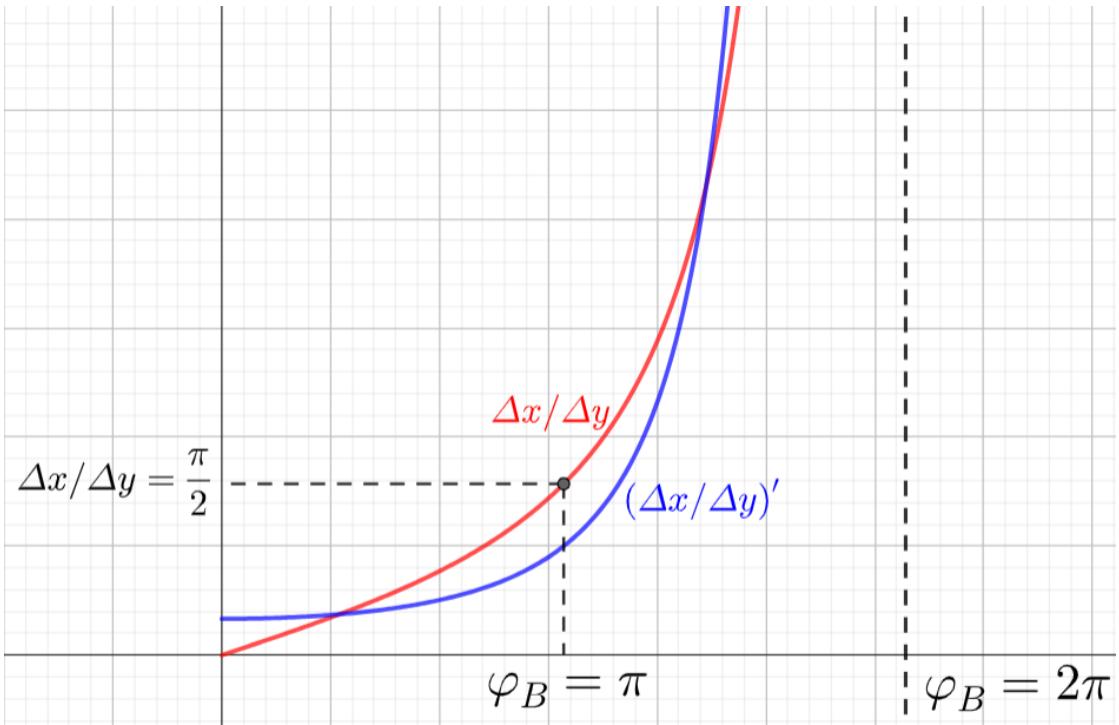


Data qualsiasi configurazione del problema della brachistocrona esteso anche a velocità iniziale non nulla, cioè per qualunque combinazione di valori di Δx , Δy e v_i , è possibile così determinare in modo univoco l'origine e il raggio della cicloide corrispondente. La curva brachistocrona soluzione del problema è il ramo di tale cicloide che collega i punti di partenza e arrivo.

Considerando il caso particolare $v_i = 0$ e $\Delta x, \Delta y > 0$, il punto di partenza A è posizionato nell'origine $X_A = 0$ e $Y_A = 0 \rightarrow \Delta x = X_B, \Delta y = Y_B$. L'unica condizione da imporre è il passaggio per B: $\Delta x = r \cdot (\varphi_B - \sin \varphi_B)$.

Si considera il rapporto $\frac{\Delta x}{\Delta y} = \frac{X_B}{Y_B} = \frac{r(\varphi_B - \sin \varphi_B)}{r(1 - \cos \varphi_B)} = \frac{(\varphi_B - \sin \varphi_B)}{(1 - \cos \varphi_B)}$.

Dal grafico della funzione $\frac{(\varphi_B - \sin \varphi_B)}{(1 - \cos \varphi_B)}$ si nota che è strettamente crescente nell'intervallo $[0, 2\pi]$. Inoltre, $\frac{\Delta x}{\Delta y} = \frac{(\varphi_B - \sin \varphi_B)}{(1 - \cos \varphi_B)} = \frac{\pi}{2}$ per $\varphi_B = \pi$ (metà periodo).



Pertanto, il valore di $\frac{\Delta x}{\Delta y}$ permette di stabilire a priori l'espressione di φ_B .

Si hanno quattro possibilità:

$$1) \frac{\Delta x}{\Delta y} < \frac{\pi}{2} \rightarrow \varphi_B < \pi \rightarrow \varphi_B \text{ è nel semiperiodo } [0, \pi] \rightarrow$$

$$\varphi_B = \cos^{-1} \left(1 - \frac{\Delta y}{r} \right).$$

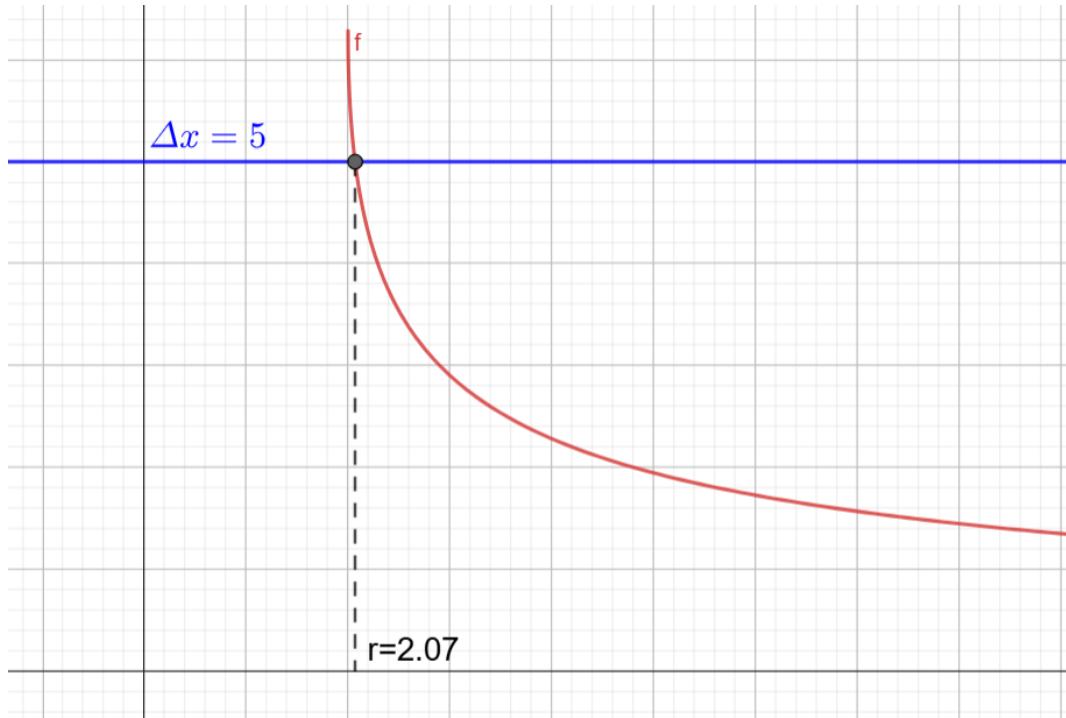
Per ottenere il raggio bisogna dunque risolvere:

$$\Delta x = r \cdot \left(\cos^{-1} \left(1 - \frac{\Delta y}{r} \right) - \sin \left(\cos^{-1} \left(1 - \frac{\Delta y}{r} \right) \right) \right).$$

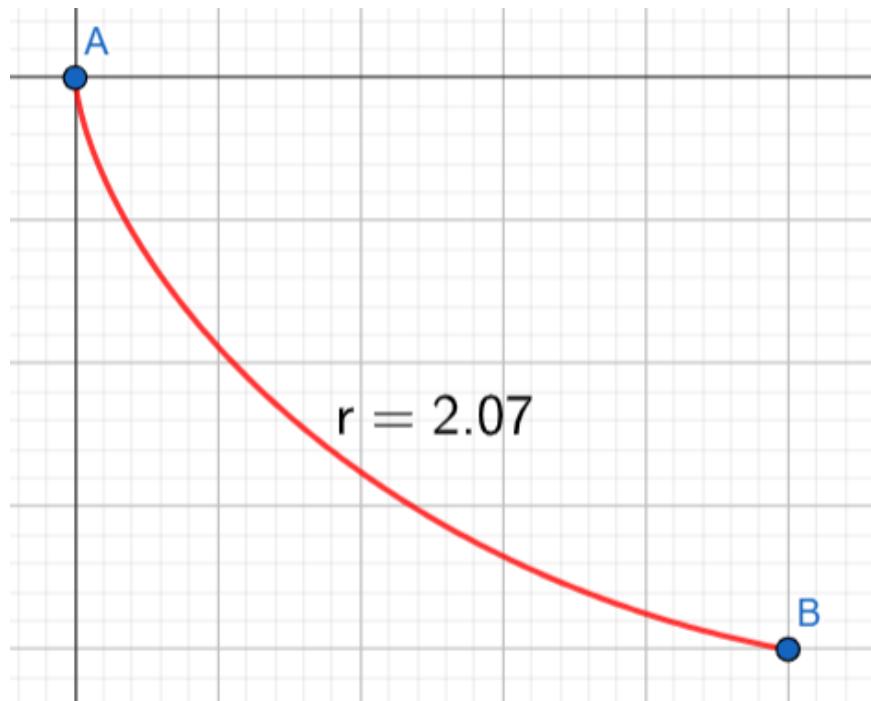
$$\Delta x = r \cdot \left(\cos^{-1} \left(1 - \frac{\Delta y}{r} \right) \right) - \sqrt{\Delta y \cdot (2r - \Delta y)}.$$

Tale equazione deve comunque essere risolta numericamente.

Esempio: $\Delta x = 5$, $\Delta y = 4$.

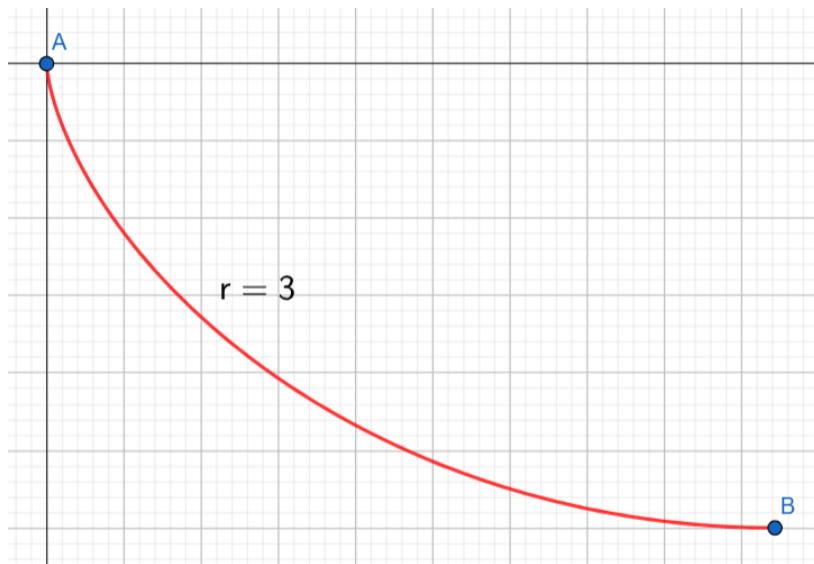


A questo punto si conosce il ramo di cicloide che collega A e B:



$$2) \frac{\Delta x}{\Delta y} = \frac{\pi}{2} \rightarrow \varphi_B = \pi \rightarrow \Delta x = r \cdot (\pi - \sin \pi) \rightarrow r = \frac{\Delta x}{\pi}.$$

Esempio: $\Delta x = 3\pi$, $\Delta y = 6 \rightarrow r = \frac{3\pi}{\pi} = 3$.



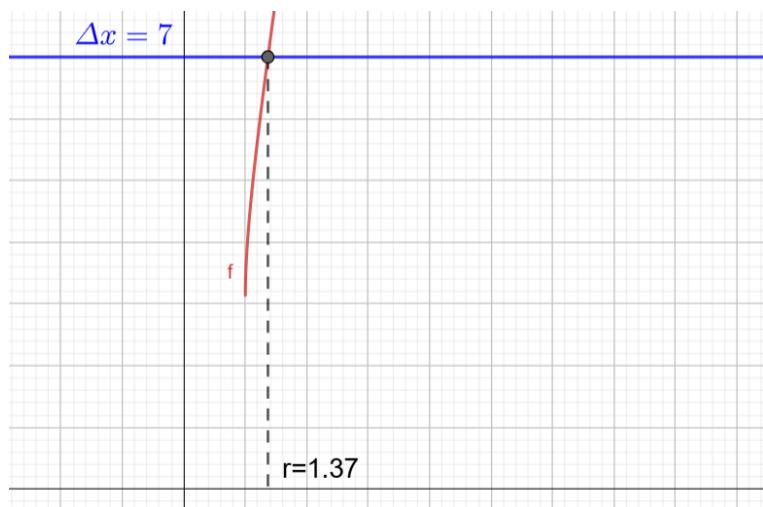
$$3) \frac{\Delta x}{\Delta y} > \frac{\pi}{2} \rightarrow \varphi_B > \pi \rightarrow \varphi_B \text{ è nel semiperiodo } [\pi, 2\pi] \rightarrow$$

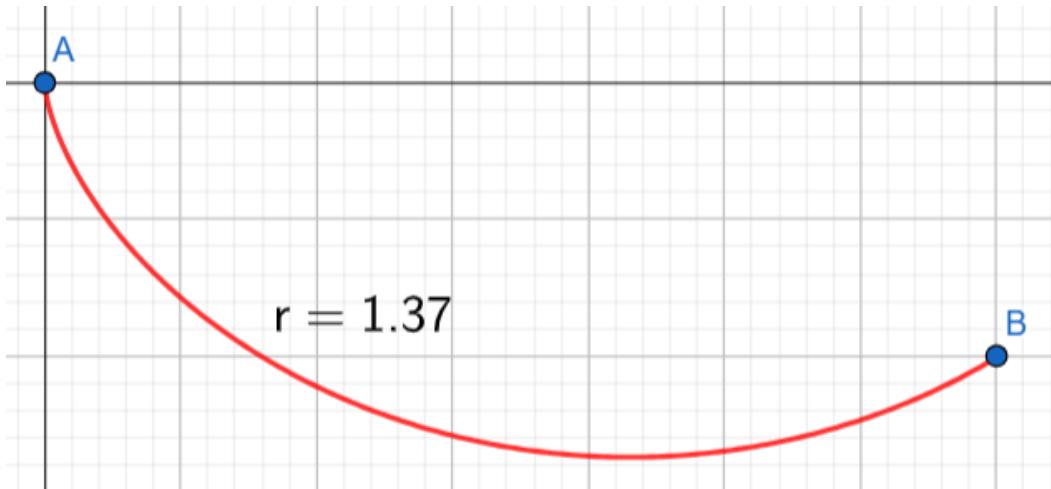
$$\varphi_B = 2\pi - \cos^{-1}\left(1 - \frac{\Delta y}{r}\right).$$

$$\Delta x = r \cdot \left(2\pi - \cos^{-1}\left(1 - \frac{\Delta y}{r}\right) - \sin\left(2\pi - \cos^{-1}\left(1 - \frac{\Delta y}{r}\right)\right)\right).$$

$$\Delta x = r \cdot \left(2\pi - \cos^{-1}\left(1 - \frac{\Delta y}{r}\right)\right) + \sqrt{\Delta y \cdot (2r - \Delta y)}.$$

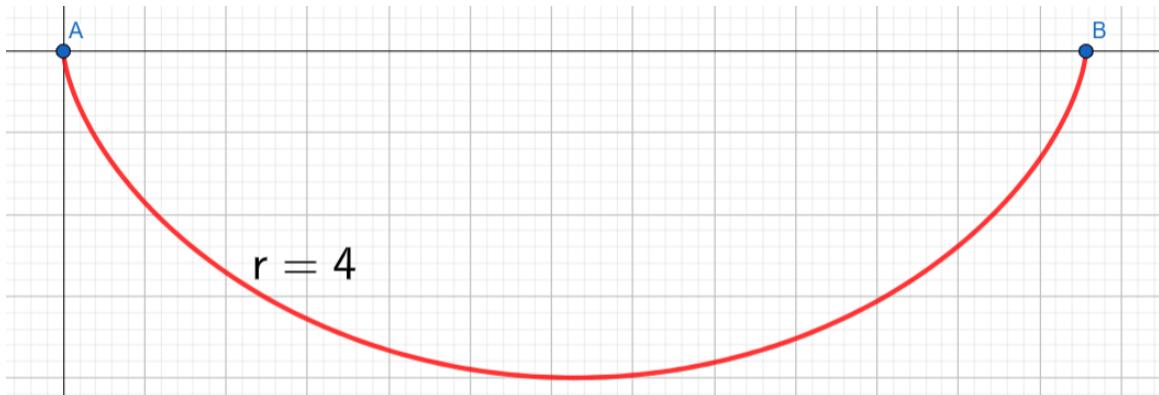
Esempio: $\Delta x = 7$, $\Delta y = 7$.





$$4) \Delta y = 0 \rightarrow \varphi_B = 2\pi \rightarrow \Delta x = r \cdot (2\pi - \sin 2\pi) \rightarrow r = \frac{\Delta x}{2\pi}.$$

$$\text{Esempio: } \Delta x = 8\pi, \Delta y = 0 \rightarrow r = \frac{8\pi}{2\pi} = 4.$$



In tutti i casi si ha pendenza verticale nel punto di partenza, di modo che la massa sia sottoposta inizialmente alla massima accelerazione, e acquisisca subito elevata velocità.

6. Soluzione con calcolo delle variazioni

La seguente soluzione è tratta da [3] e [4].

Per un' introduzione al calcolo delle variazioni si consulti "*Calcolo delle variazioni*" di G. Buttazzo, G. Dal Maso e E. De Giorgi [5].

6.1. Modello dinamico e tempo di caduta

La curva lungo cui cade la massa puntiforme si indica con $(x(t), y(t))$ per $t \in [0, T]$,

con: $x(0) = 0$, $y(0) = 0$, $x(T) = X_B$, $y(T) = Y_B \rightarrow x \in [0, X_B]$, $y \in [0, Y_B]$.

Poiché in $y = 0$ la massa parte da ferma, dalla conservazione dell'energia meccanica si ha $v^2 = 2gy$.

$$\begin{aligned} v^2 &= \left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2 = \left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dx} \cdot \frac{dx}{dt}\right)^2 = \left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dx}\right)^2 \cdot \left(\frac{dx}{dt}\right)^2 = \\ &= \left(\frac{dx}{dt}\right)^2 \cdot \left(1 + \left(\frac{dy}{dx}\right)^2\right) = 2gy. \\ \left(\frac{dx}{dt}\right)^2 &= \frac{2gy}{1 + \left(\frac{dy}{dx}\right)^2}. \end{aligned}$$

$x(t)$ è monotona crescente perché la curva deve sempre avvicinarsi a B e non può tornare indietro lungo la direzione x ; altrimenti, la massa si staccherebbe dalla curva proseguendo nel suo moto con componente della velocità orientata verso B.

Quindi $\frac{dx}{dt} > 0 \quad \forall t \in [0, T]$.

$$\frac{dx}{dt} = \sqrt{\frac{2gy}{1 + \left(\frac{dy}{dx}\right)^2}}$$

$$1 = \frac{dx}{dt} \cdot \sqrt{\frac{1 + \left(\frac{dy}{dx}\right)^2}{2gy}}.$$

Integrando rispetto al tempo in $[0, T]$:

$$\int_0^T 1 \cdot dt = \int_0^T \frac{dx}{dt} \cdot \sqrt{\frac{1 + \left(\frac{dy}{dx}\right)^2}{2gy}} dt.$$

Poiché $x(t)$ è monotona crescente, e quindi invertibile $t(x)$, nel secondo membro si può effettuare un cambio di variabile per integrare rispetto a x :

$$T = \int_{x(0)}^{x(T)} \frac{dx}{dt} \cdot \sqrt{\frac{1 + \left(\frac{dy}{dx}\right)^2}{2gy}} \cdot \frac{dt}{dx} dx = \int_0^{X_B} \sqrt{\frac{1 + \left(\frac{dy}{dx}\right)^2}{2gy}} dx.$$

6.2. Formulazione variazionale del problema

Si è così giunti a esprimere il tempo totale di caduta T come un funzionale integrale $T[y(x)]$, definito sullo spazio di funzioni $C^1([0, X_B])$ con $y > 0$ per $x \in (0, X_B]$.

L'integrandina del funzionale (lagrangiana) è $L(x, y, y') = \sqrt{\frac{1+y'^2}{y}}$ per $y > 0$.

($2g$ costituisce una costante di proporzionalità positiva e pertanto si può ignorare)

Le condizioni al bordo di Dirichlet sono $y(0) = 0$, $y(X_B) = Y_B$.

Il problema della brachistocrona si traduce quindi nel risolvere il seguente problema di calcolo delle variazioni:

$$\min \left\{ \int_0^{X_B} \sqrt{\frac{1+y'^2}{y}} \cdot dx : y \in C^1[0, X_B], y(0) = 0, y(X_B) = Y_B \right\}.$$

La funzione $y(x)$ che minimizza il funzionale rappresenta dunque la curva brachistocrona.

6.3. Equazione di Eulero-Lagrange e riduzione di Beltrami

Si procede alla risoluzione con il metodo indiretto.

Poiché L non dipende da x , si può considerare la riduzione dell'equazione di Eulero-Lagrange all'identità di Beltrami:

$$L - y' \cdot \frac{\partial L}{\partial y'} = c,$$

dove c è una costante che verrà determinata dalle condizioni al bordo e viene chiamata nei passaggi seguenti sempre c , anche se viene manipolata ma rimane comunque una costante. Si ha quindi:

$$\sqrt{\frac{1+y'^2}{y}} - \frac{y'^2}{\sqrt{y \cdot (1+y'^2)}} = c$$

$$1 + y'^2 - y'^2 = c \cdot \sqrt{y \cdot (1+y'^2)}$$

$$\sqrt{y \cdot (1+y'^2)} = c$$

$$1 + y'^2 = \frac{c}{y}$$

$$y'^2 = \frac{c}{y} - 1$$

$$y' = \pm \sqrt{\frac{c}{y} - 1}.$$

6.4. Risoluzione dell'equazione differenziale

Si è ottenuta un'equazione differenziale a variabili separabili. Separando le variabili:

$$\frac{dy}{\sqrt{\frac{c}{y} - 1}} = \pm dx$$

$$\int \frac{1}{\sqrt{\frac{c}{y} - 1}} dy = \pm \int dx$$

$$\int \frac{1}{\sqrt{\frac{c}{y} - 1}} dy = \pm x + K, \quad K \in \mathbb{R}.$$

Per eliminare la radice quadrata e riscrivere l'integranda in termini di funzioni trigonometriche si pone: $y = c(\sin z)^2$ con $z > 0$.

$$dy = 2c \cdot \sin z \cdot \cos z dz.$$

$$\begin{aligned} \int \frac{2c \cdot \sin z \cdot \cos z}{\sqrt{\frac{1}{(\sin z)^2} - 1}} dz &= \int \frac{2c \cdot \sin z \cdot \cos z}{\frac{\sqrt{1 - (\sin z)^2}}{\sin z}} dz = c \cdot \int 2 \cdot (\sin z)^2 dz = \\ &= c \cdot \int (1 - \cos 2z) dz = c \left(z - \frac{1}{2} \sin 2z \right) = \frac{c}{2} (2z - \sin 2z). \\ \pm x + K &= \frac{c}{2} (2z - \sin 2z). \end{aligned}$$

Si pone $\varphi = 2z$ ($\varphi > 0$), da cui

$$\pm x + K = \frac{c}{2} (\varphi - \sin \varphi).$$

$$y = c(\sin z)^2 = \frac{c}{2} \cdot 2(\sin z)^2 = \frac{c}{2} \cdot (1 - \cos 2z) = \frac{c}{2} \cdot (1 - \cos \varphi)$$

$$\begin{cases} \pm x = \frac{c}{2} \cdot (\varphi - \sin \varphi) + K \\ y = \frac{c}{2} \cdot (1 - \cos \varphi) \end{cases}.$$

6.5. Condizioni al bordo

Si impongono le condizioni al bordo per trovare i parametri.

$$y = 0 \rightarrow r \cdot (1 - \cos \varphi) = 0 \rightarrow \cos \varphi = 1 \rightarrow \varphi = 0.$$

$$y(0) = 0 \rightarrow x(0) = 0 \rightarrow r \cdot (-\sin 0) + K = 0 \rightarrow K = 0.$$

Dunque, $\pm x = \frac{c}{2} \cdot (\varphi - \sin \varphi)$. Poiché per $\varphi > 0$, $\frac{c}{2} \cdot (\varphi - \sin \varphi) > 0$, la soluzione corretta dati i vincoli di dominio è quella con il segno positivo per x .

Ponendo $r = \frac{c}{2}$ si riconosce l'equazione parametrica della cicloide.

$$\begin{cases} x = r \cdot (\varphi - \sin \varphi) \\ y = r \cdot (1 - \cos \varphi) \end{cases}.$$

La condizione al bordo $y(X_B) = Y_B$ permette di ottenere r univocamente.

Tale curva è una cicloide, è una funzione di x , risolve l'equazione di Eulero-Lagrange e soddisfa le condizioni al bordo, ed è quindi la candidata ad essere la soluzione del problema.

6.6. Studio della minimalità della soluzione con l'analisi della convessità della lagrangiana

Affinché la curva ottenuta sia effettivamente la brachistocrona bisogna dimostrare che è punto di minimo assoluto per il funzionale. Per fare ciò si studia la convessità della lagrangiana L .

Poiché $y > 0$ si può effettuare la trasformazione $y = \frac{1}{2}v^2$ con $v > 0$.

Tale trasformazione è biunivoca, e pertanto gli eventuali punti di minimo globali si corrispondono nella trasformazione.

Inoltre, la trasformazione è liscia e ha inversa liscia e pertanto è un diffeomorfismo. Si ha $y' = vv'$.

È così possibile studiare la convessità di L come funzione di due variabili (v, v') :

$$L(y, y') = \sqrt{\frac{1 + y'^2}{y}} \rightarrow L(v, v') = \sqrt{\frac{1 + v^2 v'^2}{\frac{1}{2}v^2}} = \sqrt{v'^2 + \frac{1}{v^2}} \cdot \sqrt{2}.$$

Ignorando il coefficiente di proporzionalità $\sqrt{2}$: $L(v, v') = \sqrt{v'^2 + v^{-2}}$.

La trasformazione rende L più semplice e regolare, eliminando l'accoppiamento non lineare tra y e y' , che può introdurre variazioni di curvatura e impedire di concludere immediatamente che la funzione sia convessa.

Con la trasformazione, invece, le variabili v e v' risultano distintamente separate, semplificando il calcolo delle derivate seconde e permettendo una verifica diretta della convessità.

Si calcolano le derivate parziali, le derivate seconde parziali e le derivate seconde miste:

$$L_v = \frac{\partial L}{\partial v} = \frac{1}{2\sqrt{v'^2 + v^{-2}}} \cdot \frac{\partial}{\partial v} (v^{-2}) = -v^{-3}L^{-1}.$$

$$L_{vv'} = \frac{\partial L}{\partial v'} = \frac{1}{2\sqrt{v'^2 + v^{-2}}} \cdot \frac{\partial}{\partial v'} (v'^2) = v'L^{-1}.$$

$$\begin{aligned} L_{v'v'} &= \frac{\partial^2 L}{\partial v'^2} = \frac{\partial L_{v'}}{\partial v'} = \frac{\partial}{\partial v'} (v' \cdot L^{-1}) = L^{-1} + v' \frac{\partial L^{-1}}{\partial v'} = L^{-1} - v'L^{-2} \frac{\partial L}{\partial v'} \\ &= L^{-1} - v'L^{-2}L_{vv'} = L^{-1} - v'^2L^{-3} \\ &= \frac{1}{\sqrt{v'^2 + v^{-2}}} - \frac{v'^2}{\sqrt{(v'^2 + v^{-2})^3}} = \frac{v'^2 + v^{-2} - v'^2}{\sqrt{(v'^2 + v^{-2})^3}} \\ &= \frac{v^{-2}}{\sqrt{(v'^2 + v^{-2})^3}} = v^{-2}L^{-3}. \end{aligned}$$

$$\begin{aligned}
L_{vv} &= \frac{\partial^2 L}{\partial v^2} = \frac{\partial L_v}{\partial v} = \frac{\partial}{\partial v} (-v^{-3}L^{-1}) = 3v^{-4}L^{-1} - v^{-3}\frac{\partial L^{-1}}{\partial v} \\
&= 3v^{-4}L^{-1} + v^{-3}L^{-2}\frac{\partial L}{\partial v} \\
&= 3v^{-4}L^{-1} + v^{-3}L^{-2}L_v = 3v^{-4}L^{-1} - v^{-6}L^{-3} \\
&= \frac{3v^{-4}}{\sqrt{v'^2 + v^{-2}}} - \frac{v^{-6}}{\sqrt{(v'^2 + v^{-2})^3}} = \frac{3v^{-4}(v'^2 + v^{-2}) - v^{-6}}{\sqrt{(v'^2 + v^{-2})^3}} \\
&= \frac{v^{-6}[3v^2(v'^2 + v^{-2}) - 1]}{\sqrt{(v'^2 + v^{-2})^3}} = \frac{v^{-6}(3v^2v'^2 + 2)}{\sqrt{(v'^2 + v^{-2})^3}} \\
&= \frac{3v^{-4}v'^2 + 2v^{-6}}{\sqrt{(v'^2 + v^{-2})^3}} = v^{-4}(3v'^2 + 2v^{-2}) \cdot L^{-3}.
\end{aligned}$$

Dato che $L \in C^2$, per il teorema di Schwarz, le derivate miste sono uguali.

$$\begin{aligned}
L_{v'v} = L_{vv'} &= \frac{\partial^2 L}{\partial v \partial v'} = \frac{\partial L_{v'}}{\partial v} = \frac{\partial}{\partial v} (v' \cdot L^{-1}) = v' \frac{\partial L^{-1}}{\partial v} = -v'L^{-2} \frac{\partial L}{\partial v} \\
&= -v'L^{-2}L_v = v^{-3}v'L^{-3}.
\end{aligned}$$

Quindi la matrice Hessiana è:

$$H_L = \begin{pmatrix} L_{vv} & L_{vv'} \\ L_{v'v} & L_{v'v'} \end{pmatrix} = \begin{pmatrix} v^{-4}(3v'^2 + 2v^{-2}) \cdot L^{-3} & v^{-3}v'L^{-3} \\ v^{-3}v'L^{-3} & v^{-2}L^{-3} \end{pmatrix}.$$

Per stabilirne la convessità si calcola il determinante:

$$\begin{aligned}
\det(H_L) &= L_{vv}L_{v'v'} - {L_{vv'}}^2 = \\
&= v^{-4}(3v'^2 + 2v^{-2}) \cdot L^{-3} \cdot v^{-2}L^{-3} - (v^{-3}v'L^{-3})^2 = \\
&= v^{-6}(3v'^2 + 2v^{-2}) \cdot L^{-6} - v^{-6}v'^2L^{-6} = \\
&= 2(vL)^{-6} \cdot (v'^2 + v^{-2}).
\end{aligned}$$

Si nota che $\det(H_L) > 0$ e $L_{vv} > 0$ e quindi l'Hessiana è definita positiva per ogni $v > 0$. Ne consegue che la lagrangiana $L \in C^2$ è strettamente convessa in tutto il

dominio di definizione e quindi anche il funzionale $T[y(x)]$ è strettamente convesso nello spazio delle funzioni $C^1[0, X_B]$.

Si può concludere che la cicloide trovata è l'unica soluzione dell'equazione di Eulero-Lagrange, ed è punto di minimo globale del funzionale. Ciò significa che si è trovata la curva brachistocrona.

7. Soluzione con teoria del controllo ottimo

La seguente soluzione è tratta dal lavoro di H. J. Sussmann e J. C. Willems [6].

Per un' introduzione alla teoria del controllo ottimo si consulti “*Optimal Control Theory: An Introduction*” di D. E. Kirk [7].

La teoria del controllo ottimo fornisce il quadro matematico più adeguato e naturale per analizzare e risolvere il problema della brachistocrona.

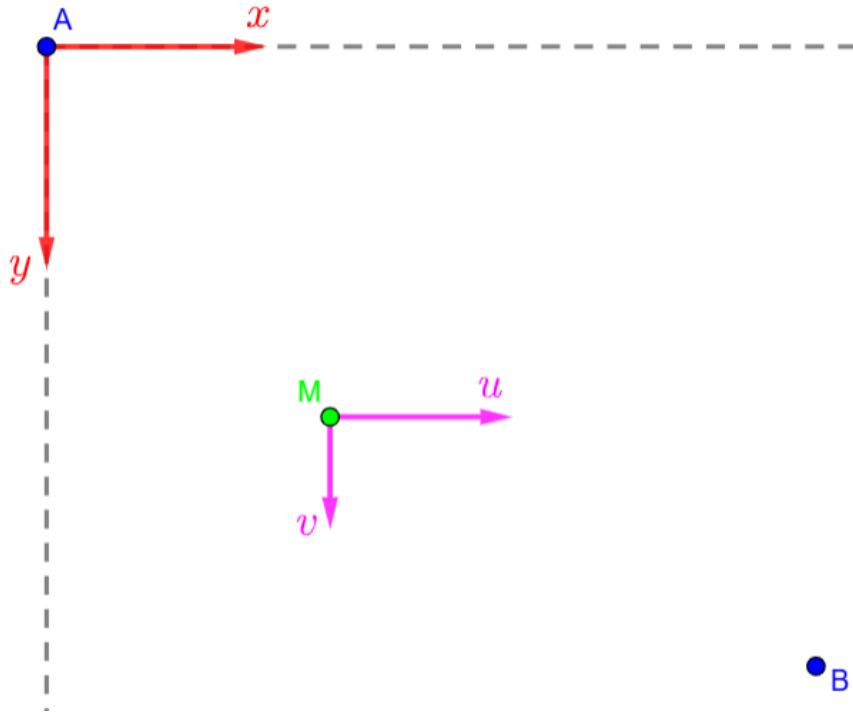
Il calcolo delle variazioni è un campo della matematica che offre metodi per minimizzare funzionali qualsiasi, indipendentemente dalla loro natura e dal loro significato. La teoria del controllo ottimo, invece, è una teoria più specifica, usata nelle applicazioni, soprattutto in ingegneria ed economia. Il suo obiettivo è quello di minimizzare funzioni di costo (funzionali integrali del tempo) associate a sistemi dinamici. La teoria del controllo ottimo, quindi, può trattare un sottoinsieme dei problemi del calcolo delle variazioni con strumenti più efficaci e mirati per tali contesti specifici.

Per il problema della brachistocrona, la teoria del controllo ottimo consente di ricavare il controllo che minimizza il tempo di caduta, e conseguentemente la traiettoria ottima determinata da tale controllo, ossia la curva brachistocrona.

Il suo principale vantaggio rispetto ad altri approcci risolutivi è che questi ultimi assumono a priori, basandosi su un'osservazione intuitiva, che la curva ottimale sia una funzione $y(x)$. Al contrario, con la teoria del controllo ottimo, questa proprietà emerge come una conseguenza dimostrata nel procedimento risolutivo.

7.1. Definizione del sistema dinamico

Il problema della brachistocrona può essere formulato come un problema di controllo ottimo: mentre nelle soluzioni precedenti la traiettoria seguita dalla massa puntiforme veniva concepita come determinata da un vincolo fisico, con la teoria del controllo ottimo si interpreta la traiettoria come determinata da un controllo che pilota la massa puntiforme. Considerando un controllo unitario, questo può solo influenzare la direzione del moto. Il suo ruolo, quindi, è del tutto equivalente a quello di un vincolo fisico.



Il moto avviene nel semipiano verticale xy per $y > 0$. Il comportamento dinamico della massa puntiforme è dato dalle seguenti equazioni differenziali del moto (ignorando la costante di proporzionalità $\sqrt{2g}$) :

$$\dot{x} = u\sqrt{y} \quad \dot{y} = v\sqrt{y}.$$

Il controllo è un vettore bidimensionale (u, v) tale che $u^2 + v^2 = 1$, in modo che può cambiare solo la direzione della velocità, ma non il suo modulo. Le sue componenti rappresentano quindi le componenti direzionali del moto.

Il modulo della velocità è invece proporzionale a \sqrt{y} , a causa dell'effetto dell'accelerazione gravitazionale che agisce sulla massa.

7.2. Hamiltoniana e principio del massimo di Pontryagin

L'Hamiltoniana del controllo ottimo è in generale:

$$H(\underline{x}, \underline{u}, \underline{p}, p_0) = \underline{p} \cdot \underline{f}(\underline{x}, \underline{u}) + p_0 L(\underline{x}, \underline{u}).$$

Dove: \underline{x} è il vettore degli stati, \underline{u} è il vettore dei controlli, \underline{p} è il vettore dei co-stati, \underline{f} è il vettore delle equazioni di stato, L è la funzione di costo istantaneo e p_0 è il

moltiplicatore associato al costo (una costante non nulla che garantisce che le soluzioni per il controllo ottimo non siano banali).

Nei problemi in cui si vuole minimizzare il tempo (problemi di tempo minimo) si sceglie $L = 1$. In questo modo il costo totale J da minimizzare, cioè l'integrale nel tempo della funzione di costo istantaneo, corrisponda al tempo totale:

$$J = \int_0^T L(\underline{x}, \underline{u}) dt = \int_0^T 1 dt = T.$$

Inoltre, si fissa $p_0 = -1$ per normalizzare la funzione di costo e riconvertire il problema di minimizzazione del tempo, in uno di massimizzazione dell'hamiltoniana, coerente con la formulazione convenzionale del principio del massimo di Pontryagin.

Per il problema della brachistocrona l'Hamiltoniana del controllo ottimo è quindi:

$$H(x, y, u, v, p, q, p_0) = (pu + qv)\sqrt{y} - 1.$$

p e q sono i co-stati rispettivamente di x e y .

Il Principio del Massimo di Pontryagin afferma che il controllo ottimo deve massimizzare l'Hamiltoniana lungo la traiettoria ottimale per ogni istante t .

7.3. Condizioni per il controllo ottimo

Se la curva $(x(t), y(t))$ per $t \in [0, T]$ è ottimale allora esistono funzioni continue $p(t)$ e $q(t)$ tali che le condizioni di massimizzazione dell'Hamiltoniana sono:

$$u(t) = \frac{p(t)}{\|\underline{p}(t)\|} \quad v(t) = \frac{q(t)}{\|\underline{p}(t)\|}.$$

Dove $\underline{p} = (p, q)$ è il vettore dei co-stati e $\|\underline{p}\| = \sqrt{p^2 + q^2}$ è la sua norma euclidea.

Questo perché per massimizzare $H = (pu + qv)\sqrt{y} - 1$ rispetto a (u, v) bisogna massimizzare il prodotto scalare $pu + qv$, e ciò si ottiene quando il controllo (u, v) è allineato con il vettore dei co-stati (p, q) . Dato che il controllo è un vettore unitario, si ottiene dunque normalizzando il vettore dei co-stati.

Poiché i co-stati evolvono nel tempo secondo le equazioni canoniche di Hamilton, la direzione ottimale del moto cambia dinamicamente. La traiettoria ottimale seguirà quindi la direzione imposta dai co-stati.

7.4. Equazioni del co-stato

Dalle equazioni canoniche di Hamilton: $\dot{p} = -\frac{\partial H}{\partial x}$ $\dot{q} = -\frac{\partial H}{\partial y}$.

Derivando l'Hamiltoniana rispetto a x : $\frac{\partial H}{\partial x} = 0 \rightarrow \dot{p} = 0$.

poiché l'Hamiltoniana non dipende esplicitamente da x (non ci sono forze lungo x che influenzano il moto).

Derivando l'Hamiltoniana rispetto a y :

$$\begin{aligned}\frac{\partial H}{\partial y} &= \frac{\partial}{\partial y} ((pu + qv)\sqrt{y} - 1) = (pu + qv) \cdot \frac{1}{2\sqrt{y}}. \\ \dot{q} &= -\frac{(pu + qv)}{2\sqrt{y}}.\end{aligned}$$

Sostituendo le espressioni per il controllo ottimo:

$$pu + qv = p \frac{p}{\|\underline{p}\|} + q \frac{q}{\|\underline{p}\|} = \frac{p^2}{\|\underline{p}\|} + \frac{q^2}{\|\underline{p}\|} = \frac{\|\underline{p}\|^2}{\|\underline{p}\|} = \|\underline{p}\|.$$

$$\text{Quindi } \dot{q} = -\frac{\|\underline{p}\|}{2\sqrt{y}}.$$

Per preservare un problema non banale di tempo minimo, si richiede $\|\underline{p}\| \neq 0$.

Si hanno così ottenuto le equazioni del co-stato (adjoint equations):

$$\begin{cases} \dot{p} = 0 \\ \dot{q} = -\frac{\|\underline{p}(t)\|}{2\sqrt{y(t)}} \end{cases}.$$

7.5. Determinazione della curva ottimale

Poiché $\dot{p} = 0$, si ha che $p = \text{costante}$.

Se $p = 0 \rightarrow u = 0 \rightarrow \dot{x} = 0$ e quindi la curva ottimale è un segmento verticale che congiunge A e B.

Invece se $p \neq 0$ si ha $\dot{x}(t) = \frac{p}{\|p\|} \sqrt{y(t)}$.

Da cui $p > 0 \rightarrow \dot{x}(t) > 0 \forall t$ e $p < 0 \rightarrow \dot{x}(t) < 0 \forall t$.

Si conclude che $x(t)$ è strettamente monotona $\forall t \in [0, T]$. Quindi è possibile parametrizzare la curva ricercata rispetto a x come $y(x)$, invece che rispetto al tempo come $(x(t), y(t))$.

Ora:

$$y' = \frac{dy}{dx} = \frac{dy}{dt} \cdot \frac{dt}{dx} = \frac{\dot{y}}{\dot{x}} = \frac{v\sqrt{y}}{u\sqrt{y}} = \frac{v}{u} = \frac{q}{p}$$

$$1 + y'^2 = 1 + \frac{q^2}{p^2} = \frac{p^2 + q^2}{p^2} = \frac{\|p\|^2}{p^2}.$$

$$y'' = \frac{1}{p} \frac{dq}{dx} = \frac{1}{p} \frac{dq}{dt} \frac{dt}{dx} = \frac{1}{p} \frac{\dot{q}}{\dot{x}}.$$

Sostituendo $\dot{x} = u\sqrt{y}$:

$$y'' = \frac{1}{p} \frac{\dot{q}}{u\sqrt{y}}.$$

Sostituendo $u = \frac{p}{\|p\|}$:

$$y'' = \frac{1}{p} \frac{\|p\| \dot{q}}{p\sqrt{y}} = \frac{\|p\| \dot{q}}{p^2 \sqrt{y}}.$$

Sostituendo $\dot{q} = -\frac{\|\underline{p}\|}{2\sqrt{y}}$:

$$y'' = -\frac{\|\underline{p}\| \|\underline{p}\|}{2\sqrt{y}\sqrt{y}p^2} = -\frac{\|\underline{p}\|^2}{2yp^2}$$

$$2yy'' = -\frac{\|\underline{p}\|^2}{p^2} = -(1 + y'^2).$$

Si ottiene infine l'equazione differenziale che deve essere soddisfatta dalla curva ottimale:

$$1 + y'^2 + 2yy'' = 0.$$

Si tratta di un'equazione non lineare, del secondo ordine, che può essere ridotta all'ordine inferiore tramite una prima integrazione.

Si pone $z = y'$ e l'equazione differenziale diventa:

$$1 + z^2 + 2y \frac{dz}{dx} = 0 \text{ con } y > 0.$$

Per poter separare le variabili si vuole esprimere z come funzione di y :

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} = \frac{dz}{dy} y' = \frac{dz}{dy} z.$$

Sostituendo nell'ODE:

$$1 + z^2 + 2yz \frac{dz}{dy} = 0$$

$$y \frac{dz}{dy} = -\frac{1 + z^2}{2z}$$

$$\frac{2z}{1 + z^2} dz = -\frac{dy}{y}.$$

Integrando entrambi i membri:

$$\int \frac{2z}{1 + z^2} dz = -\int \frac{1}{y} dy$$

$$\ln(1 + z^2) = -\ln y + K$$

$$\ln(1 + z^2) + \ln y = K$$

$$\ln(y(1 + z^2)) = K$$

$$y(1 + z^2) = e^K = c \quad (c > 0)$$

$$1 + z^2 = \frac{c}{y}$$

$$y'^2 = \frac{c}{y} - 1$$

$$y' = \pm \sqrt{\frac{c}{y} - 1}.$$

Si è ottenuta così la stessa equazione differenziale a cui si era giunti nel procedimento con il calcolo delle variazioni, la cui soluzione si è già mostrato essere la cicloide.

7.6. Verifica della soluzione

In aggiunta è possibile verificare che la cicloide soddisfa l'equazione differenziale:

$$\begin{cases} x = r \cdot (\varphi - \sin \varphi) \\ y = r \cdot (1 - \cos \varphi) \end{cases}$$

$$\begin{cases} \dot{x} = \frac{dx}{dt} = r \cdot (1 - \cos \varphi) \\ \dot{y} = \frac{dy}{dt} = r \cdot \sin \varphi \end{cases} .$$

Dalla formula di bisezione, y si può scrivere come:

$$y = r \cdot (1 - \cos \varphi) = 2r \left(\sin \frac{\varphi}{2} \right)^2.$$

Si calcola y' :

$$y' = \frac{dy}{dx} = \frac{dy}{dt} \cdot \frac{dt}{dx} = \frac{\dot{y}}{\dot{x}} = \frac{r \cdot \sin \varphi}{r \cdot (1 - \cos \varphi)} = \frac{\sin \varphi}{1 - \cos \varphi}.$$

Usando le uguaglianze trigonometriche:

$$\sin \varphi = 2 \sin \frac{\varphi}{2} \cos \frac{\varphi}{2} \quad \text{e} \quad 1 - \cos \varphi = 2 \left(\sin \frac{\varphi}{2} \right)^2.$$

$$y' = \frac{2 \sin \frac{\varphi}{2} \cos \frac{\varphi}{2}}{2 \left(\sin \frac{\varphi}{2} \right)^2} = \frac{\cos \frac{\varphi}{2}}{\sin \frac{\varphi}{2}} = \cot \frac{\varphi}{2}.$$

Si calcola y'' :

$$y'' = \frac{dy'}{dx} = \frac{dy'}{d\varphi} \cdot \frac{d\varphi}{dx} = \frac{-\frac{1}{2} \cdot \frac{1}{\left(\sin \frac{\varphi}{2} \right)^2}}{r \cdot (1 - \cos t)} = \frac{-\frac{1}{2} \cdot \frac{1}{\left(\sin \frac{\varphi}{2} \right)^2}}{r \cdot 2 \left(\sin \frac{\varphi}{2} \right)^2} = -\frac{1}{4r \left(\sin \frac{\varphi}{2} \right)^4}.$$

Sostituendo tutti i termini nell'equazione differenziale:

$$1 + \left(\cot \frac{\varphi}{2} \right)^2 + 2 \cdot 2r \left(\sin \frac{\varphi}{2} \right)^2 \cdot \left(-\frac{1}{4r \left(\sin \frac{\varphi}{2} \right)^4} \right) = 0$$

$$1 + \frac{\left(\cos \frac{\varphi}{2} \right)^2}{\left(\sin \frac{\varphi}{2} \right)^2} - \frac{1}{\left(\sin \frac{\varphi}{2} \right)^2} = 0$$

$$\frac{\left(\sin \frac{\varphi}{2} \right)^2 + \left(\cos \frac{\varphi}{2} \right)^2 - 1}{\left(\sin \frac{\varphi}{2} \right)^2} = 0$$

$$\left(\sin \frac{\varphi}{2} \right)^2 + \left(\cos \frac{\varphi}{2} \right)^2 - 1 = 1 - 1 = 0.$$

Quindi, si ha verificato direttamente che la parametrizzazione della cicloide soddisfa l'equazione differenziale.

8. Simulazione software caduta di gravi lungo curve

Dopo aver analizzato il problema della brachistocrona attraverso diversi approcci teorici, è interessante estendere lo studio alla realizzazione di una simulazione software che permetta di visualizzare e confrontare il moto di caduta di gravi lungo diverse curve, sotto l'effetto della sola forza di gravità. Questo progetto costituisce un'indagine computazionale del problema, finalizzata a fornire una dimostrazione numerica della correttezza delle soluzioni teoriche, dando prova tangibile che la cicloide è effettivamente la curva che minimizza il tempo di caduta in ogni configurazione di punto iniziale e finale.

In questo modo, la simulazione si pone come un ponte tra teoria matematica e applicazione informatica, dimostrando l'efficacia dell'approccio computazionale nello studio dei problemi di ottimizzazione in fisica e ingegneria.

L'obiettivo è coniugare un rigoroso studio simulativo con un utilizzo semplice e intuitivo, adatto anche agli utenti meno esperti, offrendo una rappresentazione visiva del fenomeno e facilitando la comprensione del problema.

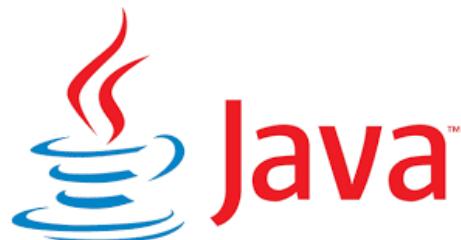
Il progetto è stato chiamato **FallSimulator**.



Il codice e la documentazione sono disponibili in una repository pubblica su GitHub al seguente link: <https://github.com/MarcoMarcoulli/FallSimulatorRepository>.

8.1. Paradigma di programmazione e strumenti utilizzati

Il progetto è stato sviluppato secondo il paradigma della programmazione orientata agli oggetti (OOP) e implementato in Java.



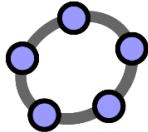
L'interfaccia grafica è stata realizzata con il framework JavaFX, scelto per garantire una UI moderna, reattiva ed efficiente, in particolare per la gestione dell'animazione della caduta.

Per la progettazione del sistema è stata adottata la modellazione UML, con la creazione di diversi diagrammi finalizzati a rappresentare la struttura e il funzionamento del software.

Gli strumenti utilizzati per lo sviluppo includono:

- **GitHub** per il controllo di versione e la gestione della configurazione.
- **Eclipse** come ambiente di sviluppo integrato (IDE).
- **Papyrus** per la realizzazione dei diagrammi UML.
- **StanIDE** per l'analisi della struttura modulare e la valutazione delle metriche di complessità.
- **SonarLint** per l'analisi statica della qualità del codice.

- **GeoGebra**



per la progettazione matematica.

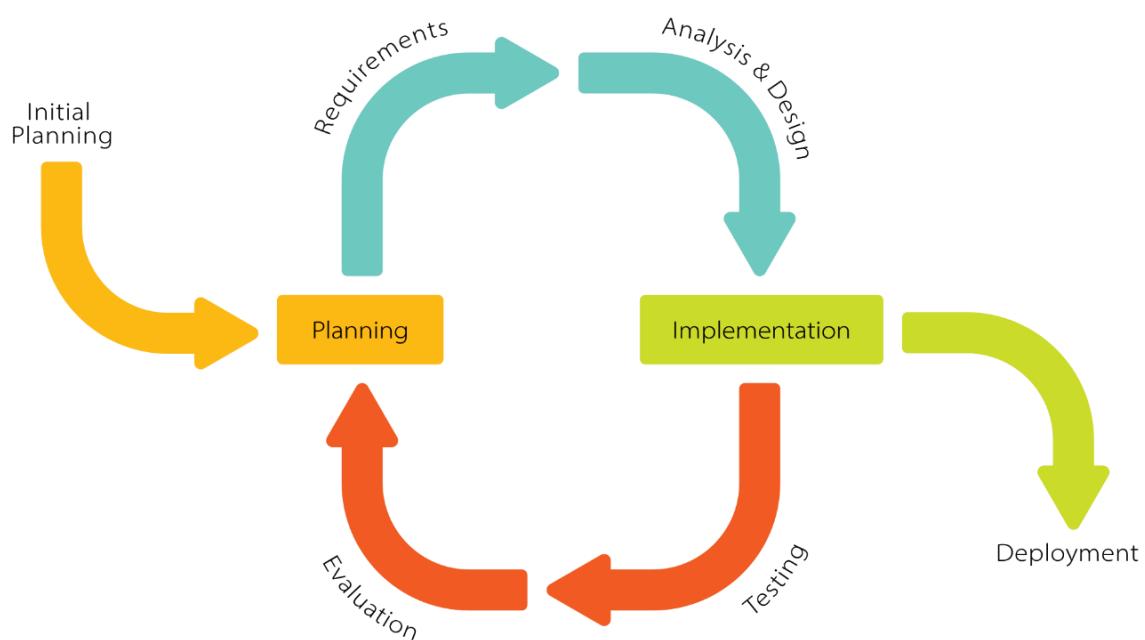
8.2. Modello di processo di sviluppo del software

Il progetto è stato realizzato con l'approccio della prototipazione evolutiva, a causa dell'elevata incertezza sull'esito dello sviluppo dovuta alla mancanza di esperienza nella realizzazione di software di tale complessità.

Si è cercato di ottenere quanto prima un sistema funzionante e successivamente è stata svolta una corposa attività di ristrutturazione, per migliorare la qualità del codice e correggere bug.

L'approccio della prototipazione evolutiva ha permesso di validare rapidamente le scelte progettuali, testando l'interazione con l'interfaccia grafica e il comportamento del sistema, prima di consolidare l'architettura finale.

Nel contesto dello sviluppo del prototipo è stato seguito il modello dello sviluppo incrementale, partendo subito con un sistema funzionante e aggiungendo le funzionalità gradualmente.



Tale scelta è stata spinta dalla caratteristica del sistema di essere organizzato secondo un rigido schema di interazione con l'utente, e pertanto la funzionalità è stata aggiunta seguendo l'ordinamento temporale di tale flusso di interazione.

8.3. Requisiti

I requisiti sono stati suggeriti dal contesto del problema della brachistocrona.

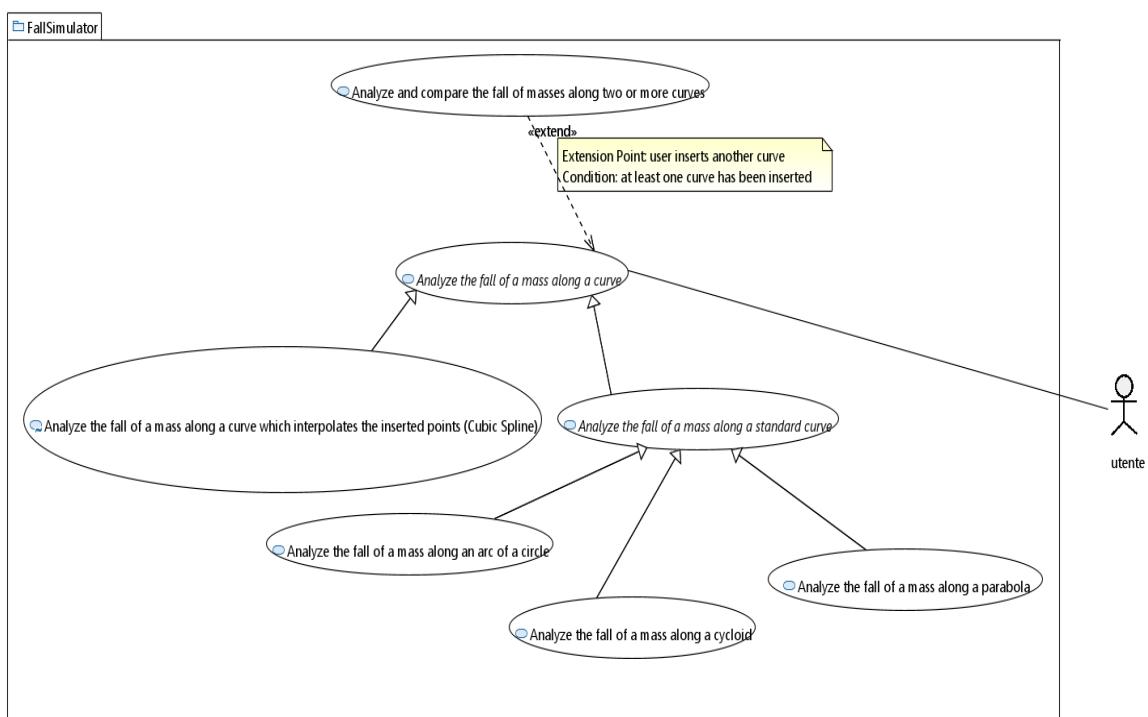
Non è stata necessaria una corposa fase di ingegneria dei requisiti in quanto la complessità del sistema non è dovuta alla complessità della modellazione dell'UoD, quanto piuttosto dalla complessità intrinseca dei componenti.

La fase dei requisiti ha riguardato principalmente le decisioni su come dovesse essere realizzata l'interfaccia e l'interazione con l'utente.

Si è optato per un approccio caratterizzato da un'interfaccia pulita, con pochi elementi e un'interazione con l'utente semplice, veloce, e finalizzata all'obiettivo, senza richiedere lunghe e ingombranti configurazioni.

Si ha deciso di conferire all'applicazione un'enfasi simpatica e graficamente accattivante, per rendere l'utilizzo piacevole e ricreativo.

I requisiti funzionali sono descritti nello use case diagram UML:

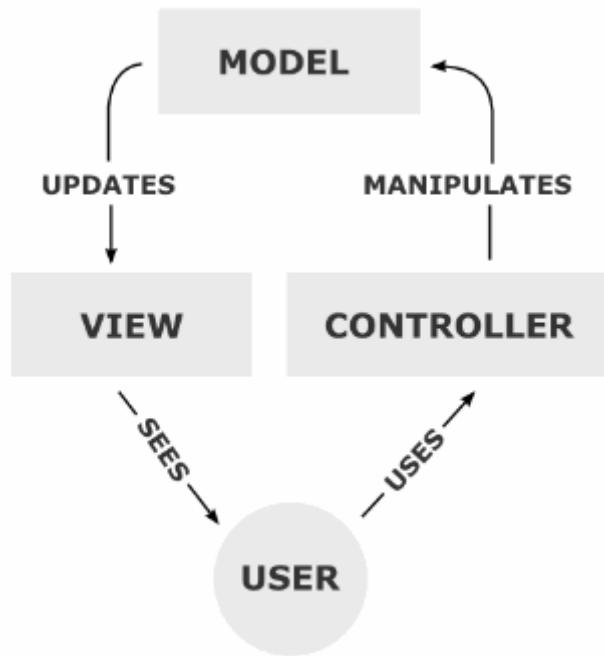


8.4. Architettura del software

Non è stata eseguita un' approfondita fase architettonica. La qualità è stata incorporata durante lo sviluppo con miglioramenti suggeriti dal test manuale.

Il sistema segue uno stile architetturale che si ispira a Model-View-Controller (MVC):

- Il **Model** gestisce i calcoli e il popolamento delle strutture dati in background.
- Il **View** si occupa della rappresentazione grafica dell'interfaccia utente.
- Il **Controller** valida gli input dell'utente e trasmette i comandi al Model.



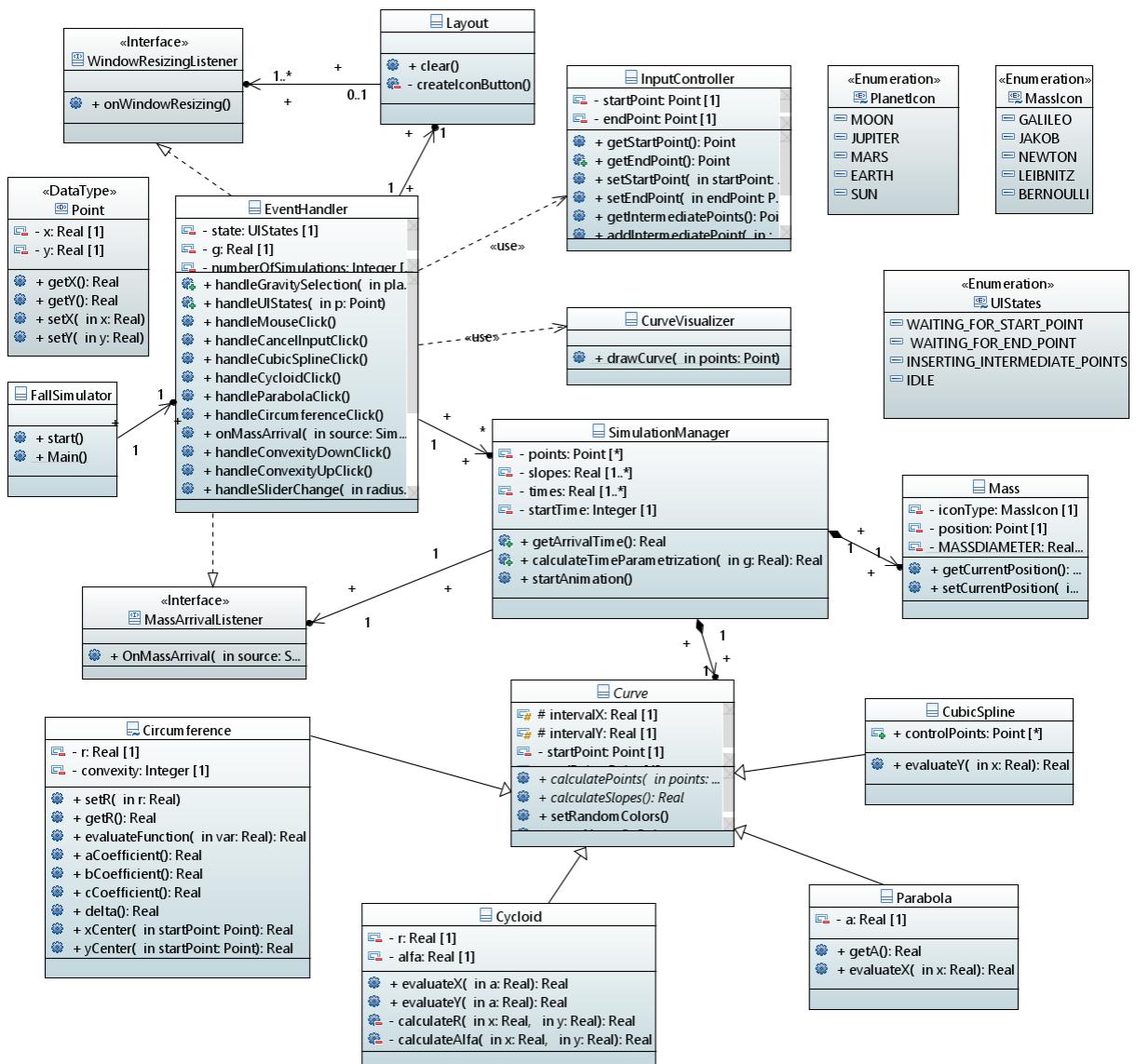
Per la realizzazione dell'interfaccia interattiva è stato seguito lo stile architetturale dell'invocazione implicita, con la definizione di gestori di eventi.

8.5. Progettazione

La fase di progettazione è stata svolta utilizzando la modellazione UML, con l'elaborazione di diversi diagrammi per rappresentare la struttura e il comportamento del sistema:

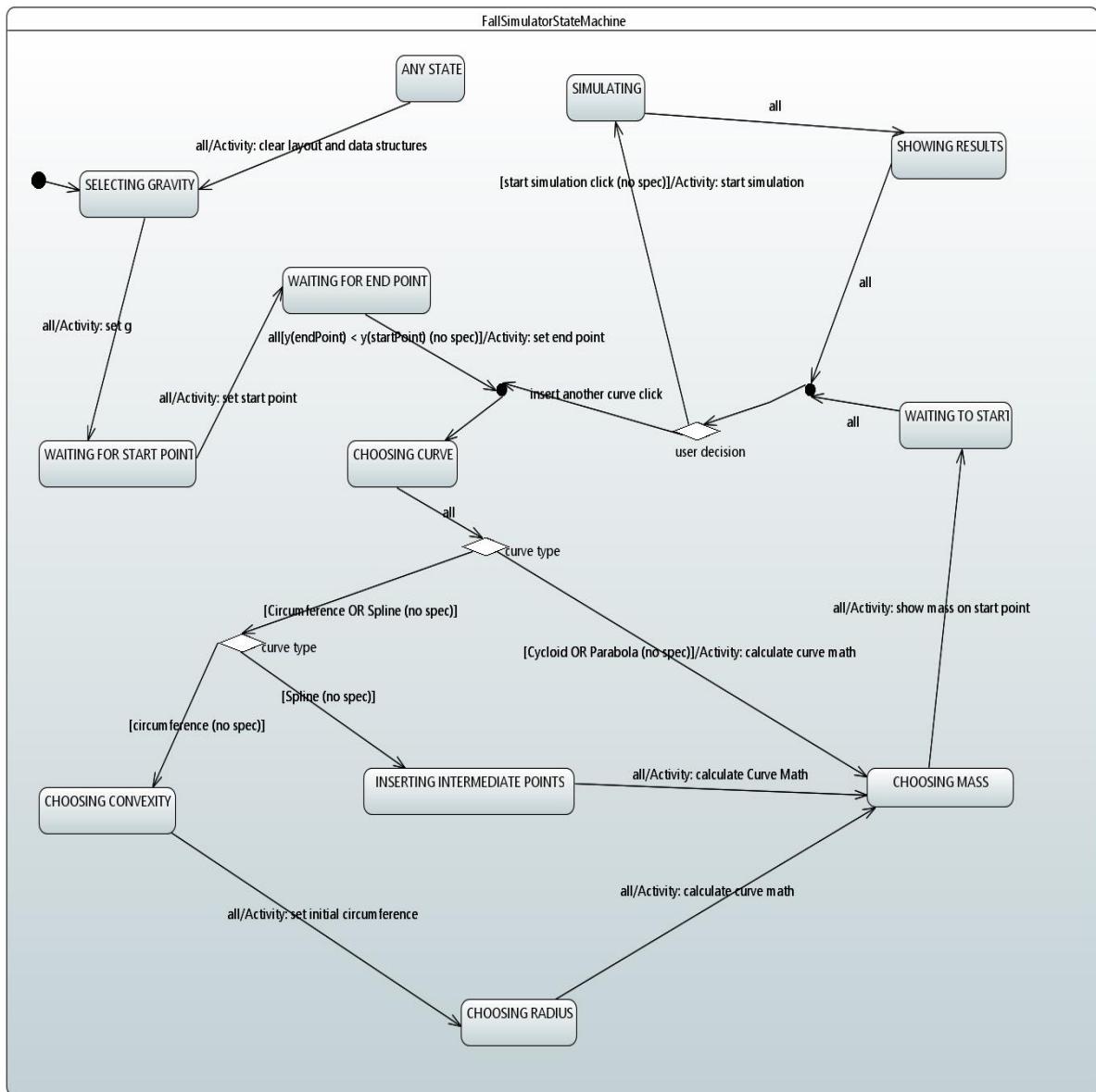
- Class diagram

È stato usato come modello per un'attività di model driven architecture (MDA), costruendo lo scheletro del sistema con il tool di generazione automatica del codice di Papyrus. Durante la ristrutturazione è stato ampiamente modificato per migliorare la manutenibilità e modularità del codice.



- State machine diagram

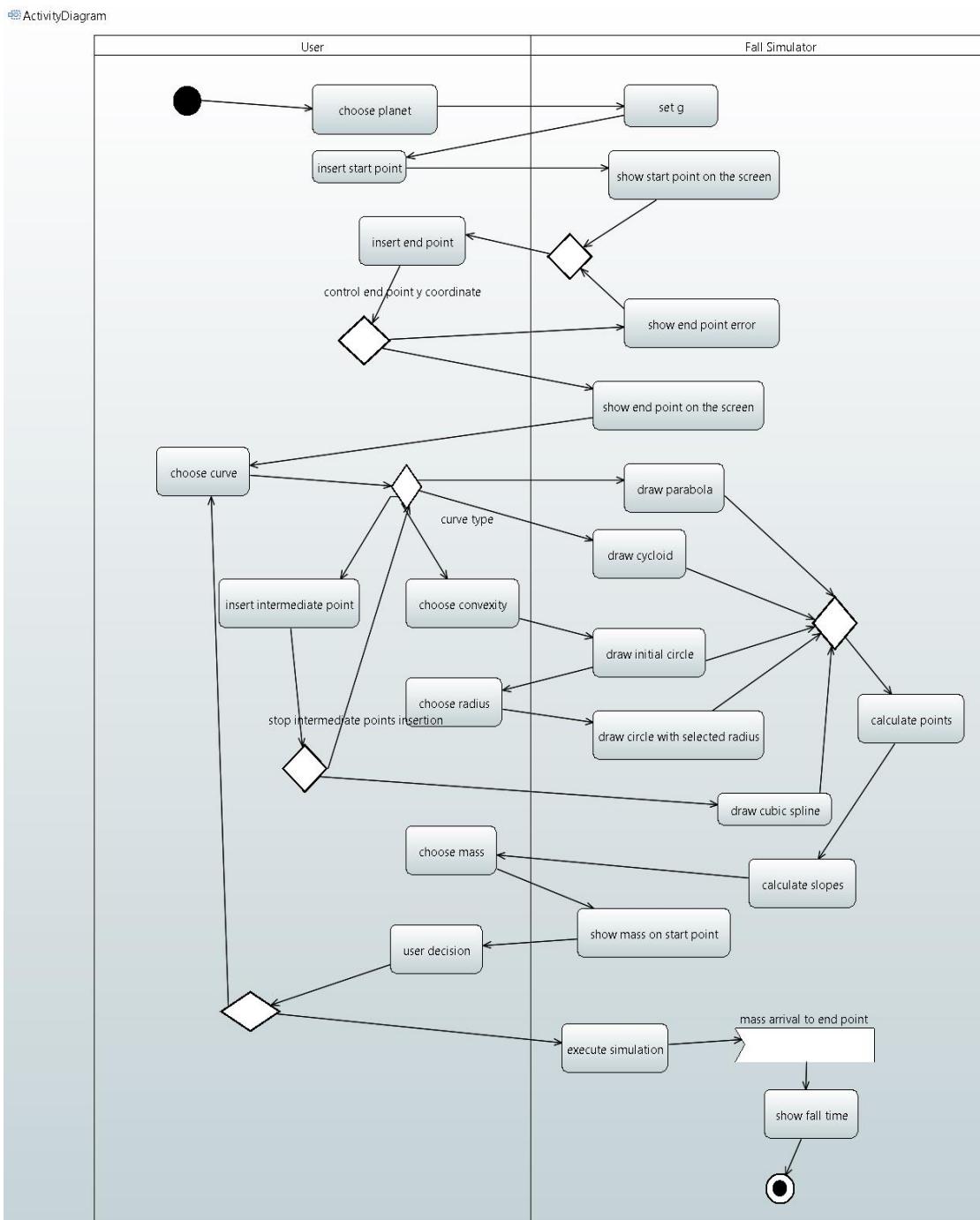
È stato definito in una fase iniziale per chiarire il flusso dell'interazione con l'utente ed è stato utilizzato per guidare lo sviluppo. La sua struttura è rimasta inalterata per tutto il ciclo di vita.



I rimanenti diagrammi sono stati eseguiti successivamente allo sviluppo del codice e sono serviti ad esplicitare graficamente la struttura e il comportamento del sistema.

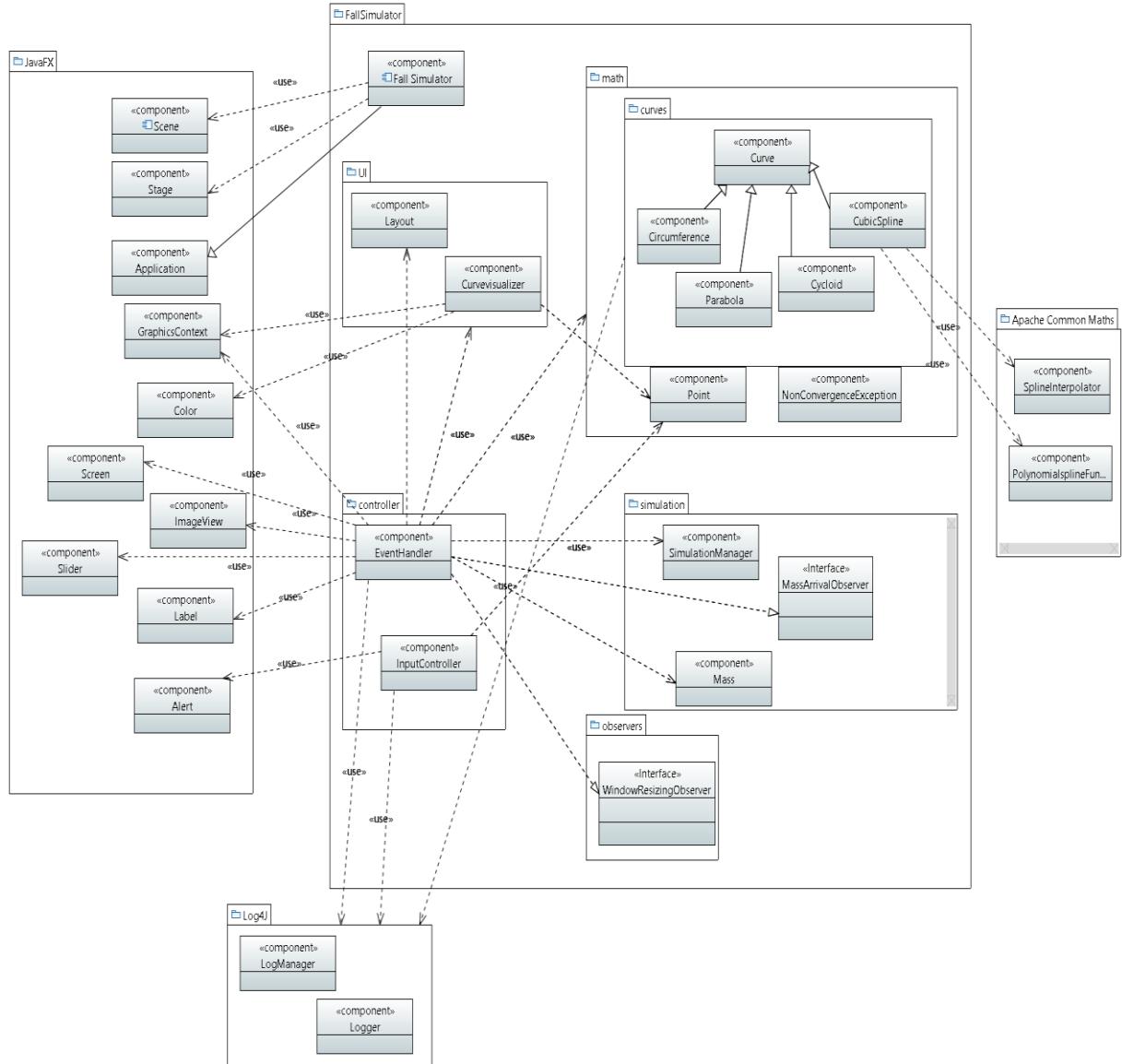
- **Activity diagram**

Modella il comportamento dinamico del sistema, rappresentando l'interazione dell'utente e il funzionamento operativo del software. Descrive il flusso di esecuzione delle operazioni, evidenziando le decisioni condizionali, e le dipendenze tra i processi.



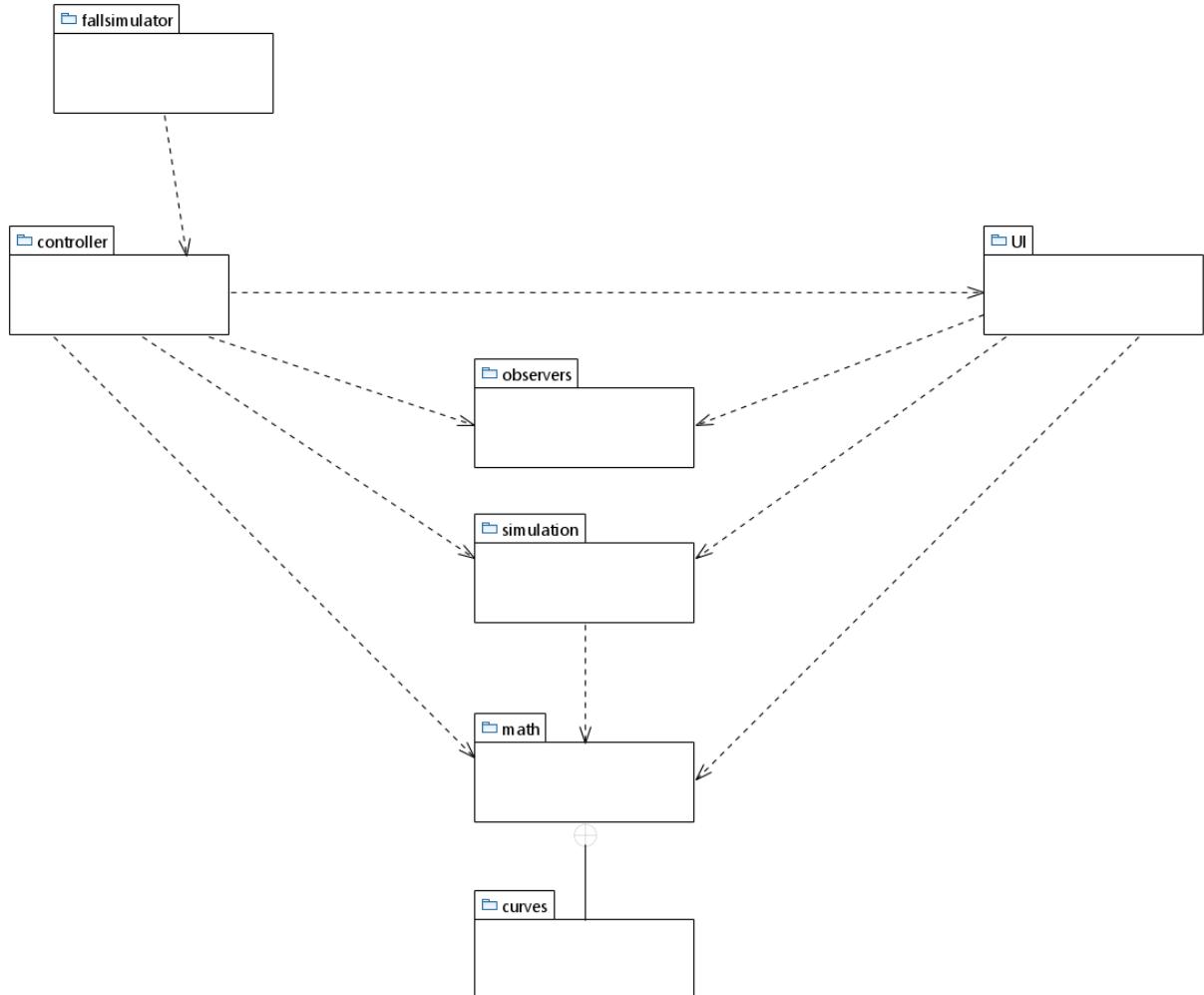
- Component diagram

Esplicita la struttura statica del sistema, rappresentando la suddivisione del software in componenti e le relazioni tra di essi, oltre che le dipendenze da librerie e framework esterni. Fornisce una visione dettagliata dell'organizzazione del codice e del modo in cui i moduli comunicano tra loro attraverso interfacce e connessioni, supportando la scalabilità e la manutenibilità del sistema.



- Package diagram

Fornisce una visione globale e di alto livello del sistema, rappresentando l'organizzazione del software in moduli che raggruppano componenti logicamente affini. Descrive la suddivisione in pacchetti, le dipendenze tra di essi e le relazioni gerarchiche. I pacchetti costituiscono unità riutilizzabili, favorendo modularità e manutenibilità del codice.



8.6. Design pattern

Sono state adottate due tipologie di design pattern che consentono di strutturare il codice in modo più chiaro ed estensibile, facilitando l'interazione tra i componenti e ottimizzando la gestione degli eventi:

- Singleton pattern per garantire un'unica istanza della classi `Layout` e `EventHandler`, responsabili rispettivamente della gestione dell'interfaccia grafica e degli eventi.

Singleton per `Layout`:

```
private static Layout theLayout = null; //singleton instance

private Layout() //private constructor

//singleton get method
public static Layout getLayout(){
    if(theLayout == null)
        theLayout = new Layout();
    return theLayout;
}
```

Singleton per `EventHandler`:

```
private static EventHandler theHandler = null; //singleton instance

private EventHandler() //private constructor

//singleton get method
public static EventHandler getHandler(){
    if(theHandler == null)
        theHandler = new EventHandler();
    return theHandler;
}
```

- Observer pattern per reagire all'arrivo del grave, visualizzando il relativo tempo di caduta e per reagire a cambiamenti della dimensione della finestra, cancellando l'input inserito e tornando alla situazione iniziale.

Definizione interfaccia per l'observer dei cambiamenti della dimensione della finestra:

```
//interface for observer that reacts when the window changes dimensions
public interface WindowResizingObserver{
    void onWindowResizing();
}
```

EventHandler implementa l'interfaccia WindowResizingObserver e aggiunge sé stesso come WindowResizingObserver di Layout e definisce il metodo onWindowResizing dell'interfaccia:

```
public class EventHandler implements WindowResizingObserver

private Layout layout;

//EventHandler observes layout
layout.addWindowResizingObserver(this);

//WindowResizing method implementation
@Override
public void onWindowResizing(){
    //calls cancel input when window gets resized
    handleCancelInputClick();
}
```

Layout ha un WindowResizingObserver come campo e dunque implementa i metodi per aggiungere e notificare l'observer:

```
//observer for handling window resizing
private WindowResizingObserver observer;

public void addWindowResizingObserver(WindowResizingObserver observer){
    this.observer=observer;
}

//notify method
private void notifyWindowResizingObserver(){
    observer.onWindowResizing();
}
```

Layout notifica WindowResizingObserver quando cambiano le dimensioni della finestra:

```
//event that calls WindowResizingObserver
root.widthProperty().addListener((obs, oldVal, newVal) -> {
    double newWidth = newVal.doubleValue() - controlPanel.getWidth();
    curveCanvas.setWidth(newWidth);
    pointsCanvas.setWidth(newWidth);
    animationPane.setPrefWidth(newWidth);
    notifyWindowResizingObserver();
});
```

Definizione interfaccia per l'observer dell'arrivo dei gravi:

```
//interface for observer that reacts when the mass stops its motion
public interface MassArrivalObserver {
    //method for handling mass arrival to endpoint (arrived = true) or mass
    //stopping along the path (arrived = false)
    void onMassArrival(SimulationManager source, boolean arrived);
}
```

EventHandler implementa l'interfaccia MassArrivalObserver, aggiunge sé stesso come MassArrivalObserver di SimulationManager e definisce il metodo onMassArrival dell'interfaccia:

```
public class EventHandler implements MassArrivalObserver

private List<SimulationManager> simulations;

//get last simulation from simulations list
SimulationManager lastSimulation(){
    return simulations.get(simulations.size()-1);
}

lastSimulation().addMassArrivalObserver(this);

//MassArrival method implementation
@Override
public void onMassArrival(SimulationManager source, boolean arrived) {
    //decrease the number of simulations running
    number_of_simulations--;
    //index of the simulation that has arrived
    int i = simulations.indexOf(source);
    if(i>=0){
        if(arrived) {//if the mass has reached end point
            layout.getMassArrivalMessagesBox().getChildren().removeAll(layout.getArrivalTimeMessages());
            //get arrived mass name string
            String massName =
                simulations.get(i).getMass().getIconTypeString();
            //get the time taken to reach end point in string format
            String arrivalTime = String.format("%.5f",
                simulations.get(i).getArrivalTime());
            //get curve name string
            String curveName = simulations.get(i).getCurve().curveName();
            //add arrival message for current simulation
            layout.getArrivalTimeMessages().add(new Label(massName + " sulla " + curveName + " è arrivato in " + arrivalTime + " secondi."));
            //order arrival times messages in increasing order respect to
            //arrival times
            layout.getArrivalTimeMessages().sort
```

```

        (Comparator.comparingInt(label ->
            extractNumber(label.getText())));
        layout.getMassArrivalMessagesBox().getChildren().addAll(0,
            layout.getArrivalTimeMessages());
    }
    //if the mass stopped before end point
    else{
        layout.getMassArrivalMessagesBox().getChildren().removeAll(layout
            .getNeverArriveMessages());
        //get arrived mass name string
        String massName =
            simulations.get(i).getMass().getIconTypeString();
        //get curve name string
        String curveName = simulations.get(i).getCurve().curveName();
        //add never arrive message for current simulation
        layout.getNeverArriveMessages().add(new Label(massName + " sulla"
            + curveName + " non arriverà mai a destinazione"));

        layout.getMassArrivalMessagesBox().getChildren().addAll(
            layout.getNeverArriveMessages());
    }
}
//if all the simulations has arrived
if(numberOfSimulations == 0)
{
    layout.getControlPanel().getChildren().clear();
    //if there are no more masse to choose it forces to start simulation
    if(layout.getMassIconButtons().getChildren().isEmpty())
        layout.getControlPanel().getChildren().addAll(layout.getBtnStartSimula
        tion(), layout.getBtnCancelInput(),
        layout.getMassArrivalMessagesBox());

    //if there are more masses it's possible to add another curve
    else
        layout.getControlPanel().getChildren().addAll(layout.getBtnStartSimula
        tion(), layout.getBtnInsertAnotherCurve(), layout.getBtnCancelInput(),
        layout.getMassArrivalMessagesBox());
}
}

```

SimulationManager ha un MassArrivalObserver come campo e dunque implementa i metodi per aggiungere e notificare l'observer:

```

//observer for handling mass arrival
private MassArrivalObserver observer;

public void addMassArrivalObserver(MassArrivalObserver observer){
    this.observer = observer;
}

```

```

//notify method
public void notifyMassArrivalObserver(boolean arrived){
    observer.onMassArrival(this, arrived);
}

```

SimulationManager notifica MassArrivalObserver quando il grave arriva al punto di arrivo o raggiunge la quota del punto di partenza, non potendo più proseguire poiché ha esaurito l'energia cinetica:

```

//timer to manage simulation
timer = new AnimationTimer() {
    //define timer behaviour
    @Override
    public void handle(long now) {
        if (startTime == 0)
            //initialize start time to current time
            startTime = now;
        // time passed from start time in seconds
        double elapsedTime = (now - startTime) / 1_000_000_000.0;
        // find closest index of times array using binary search
        int index = Arrays.binarySearch(times, elapsedTime);
        if (index < 0) {
            // If there is not an exact correspondence, `binarySearch` returns
            -(insertionIndex) - 1
            index = -index - 1; //get index of previous element respect to
            where the time would be in the ordered array
        }
        //arrival to end point handling for the mass to position exactly
        on end point
        if (index >= times.length - 1) {
            double newX = points[points.length - 1].getX() -
                mass.getMassDiameter() / 2;
            double newY = points[points.length - 1].getY() -
                mass.getMassDiameter() / 2;
            mass.getIcon().relocate(newX, newY);
            this.stop();
            notifyMassArrivalObserver(true); //arrived=true
            return;
        }
        //handling for the case in which the mass reach a point with the
        same y of start point, so it needs to stop on that point
        if (index < times.length - 2 && points[index + 2].getY() <
            points[0].getY()) {
            mass.getIcon().relocate(points[index].getX() -
                mass.getMassDiameter() / 2,
                points[index].getY() - mass.getMassDiameter() / 2);
            this.stop();
            notifyMassArrivalObserver(false); //arrived=false
            return;
        }
    }
}

```

```

//calculate mean position between points[index] and points[index
+ 1] weighted respect to proximity of elapsed time to times[index]
and times[index+1]
double ratio = (elapsedTime - times[index]) / (times[index + 1] -
times[index]);
double x = points[index].getX() + (points[index + 1].getX() -
points[index].getX()) * ratio - mass.getMassDiameter() / 2;
double y = points[index].getY() + (points[index + 1].getY() -
points[index].getY()) * ratio - mass.getMassDiameter() / 2;

// set mass position
mass.getIcon().relocate(x, y);
};

```

8.7. Implementazione

Per la scrittura del codice è stato usato l'ambiente di sviluppo Eclipse.

Per la gestione delle dipendenze esterne e l'automazione della build del progetto è stato adottato Maven. Le principali librerie e framework definite nel pom.xml di Maven e utilizzate nel codice sono:

- JavaFX  per l'interfaccia grafica.
- Log4J  per il logging e la tracciabilità degli errori.
- Apache Commons Math  per l'interpolazione spline.
- Junit 4  per il testing d'unità di classi e metodi.

Si descrive l'implementazione della rappresentazione delle curve:

Sono stati definiti quattro tipi di curve: parabola, cicloide, circonferenza con convessità verso l'alto o verso il basso e spline cubica, per consentire all'utente di delineare una traiettoria desiderata inserendo semplicemente dei punti da interpolare. Questa scelta è stata adottata per garantire un'interazione semplice e intuitiva, evitando configurazioni complesse come l'inserimento manuale delle equazioni delle curve.

Nella simulazione le curve sono implementate come vincoli bilateri, cioè come se fossero binari lungo cui la massa è forzata a seguire la traiettoria. La formulazione classica del problema della brachistocrona considera invece le curve come vincoli unilateri, cioè come guide su cui una massa scivola vincolata soltanto inferiormente. Questa è la ragione per cui nel problema si assume che la curva brachistocrona debba essere una funzione di x e non possa dunque invertire direzione lungo x . Nella simulazione invece, essendo le curve vincoli bilateri, è concessa l'inversione della direzione lungo x . In particolare, ciò può accadere per la circonferenza con convessità verso il basso.

Le curve sono state implementate con apposite classi che ereditano dalla superclasse astratta Curve, rappresentante una curva generica.

8.7.1. Curve

Definizione classe astratta Curve e dichiarazione dei campi:

```
public abstract class Curve{
    //intervals domains
    protected double intervalX;
    protected double intervalY;
    //endpoints
    protected Point startPoint;
    protected Point endPoint
    //color components
    protected int red;
    protected int green;
    protected int blue;
    //number of points of the curve
    protected static final int NUMPOINTS = 7000;
```

Vengono definiti gli attributi generici delle curve come gli intervalli di definizione in x e y , i punti di partenza e arrivo, le componenti RGB del colore e, infine, il numero di punti usati per rappresentare la curva. Sono tutti dichiarati `protected` perché devono essere accessibili dalle sottoclassi concrete. Il numero di punti è definito come un campo statico perché è comune a tutte le curve, inoltre si è deciso euristicamente di impostarlo a 7000 perché si ha notato un buon compromesso tra prestazioni e accuratezza; quindi, lo si è definito `final` per non poterlo modificare.

Definizione metodi astratti da implementare nelle sottoclassi concrete:

```
//returns the array of the points of the curve  
public abstract Point[] calculatePoints();  
//returns the array of the slopes in each point of the curve  
public abstract double[] calculateSlopes();  
//returns the name of the curve  
public abstract String curveName();
```

Vengono dichiarati metodi astratti per funzionalità generali, ma la cui implementazione è specifica di ogni tipo di curva: calcolare l'array di punti per rappresentare la curva, calcolare l'array delle pendenze (derivate prime) della curva in ogni punto e restituire il nome del tipo di curva.

Costruttore:

```
protected Curve(Point startPoint, Point endPoint){  
    this.startPoint = startPoint;  
    this.endPoint = endPoint;  
    intervalX = endPoint.getX() - startPoint.getX();  
    intervalY = endPoint.getY() - startPoint.getY();  
}
```

Viene definito un primo costruttore per inizializzare i punti di partenza e arrivo con i parametri e calcolare gli intervalli di definizione come differenza tra le componenti x e y dei punti di partenza e arrivo.

In JavaFX l'origine del sistema di coordinate è posizionata in alto a sinistra della finestra, l'asse x cresce da sinistra a destra (come nel sistema cartesiano standard), ma l'asse y cresce dall'alto verso il basso, (opposto al sistema cartesiano standard).

Il metodo `setEndpoint(Point endPoint)` di `InputController` verifica durante l'inserimento da parte dell'utente che il punto di arrivo non sia più in alto del punto di partenza. La condizione `endPoint.getY() >= startPoint.getY()` deve essere soddisfatta e di conseguenza `intervalY >= 0`. `intervalX` invece può essere sia positivo che negativo ma non può essere uguale a 0, perché sempre il metodo `setEndpoint(Point endPoint)` di `InputController` verifica i punti di partenza e arrivo non siano esattamente sulla stessa verticale, per evitare situazioni degeneri, e, nel caso lo fossero, sposta automaticamente il punto di arrivo di un pixel, di modo che `intervalX = 1`.

Metodo per cui viene già data un'implementazione in Curve:

```
Random random = new Random();
//random color setter for the curve
public void setRandomColors(){
    //selects a random integer value for the color component
    red = random.nextInt(230);
    blue = random.nextInt(230);
    green = random.nextInt(230);
}
```

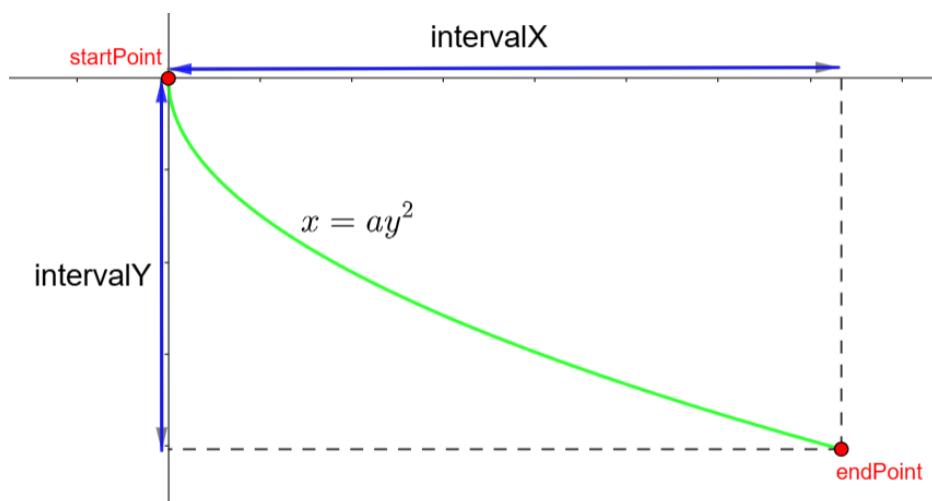
Viene definito un metodo per impostare in modo casuale le componenti RGB di colore della curva. Si ha impostato la selezione di un intero casuale tra 0 e 230 invece che tra 0 e 255 per evitare che il colore sia troppo chiaro, risultando poco evidente rispetto allo sfondo bianco.

8.7.2. Parabola

Definizione classe Parabola e dichiarazione dei campi:

```
public class Parabola extends Curve {
    //quadratic coefficient
    private double a;
```

Si vuole rappresentare un ramo di parabola con vertice nel punto di partenza e simmetrica rispetto all'asse x . Per semplificare i calcoli, il punto di partenza viene considerato inizialmente come l'origine del sistema di coordinate. La parabola è quindi descritta dall'equazione $x = ay^2$ e, il coefficiente del termine quadratico a , basta a determinare la curva in modo univoco. Solo al termine del calcolo dei punti, verrà applicata una traslazione per esprimere le coordinate nella loro posizione assoluta, tenendo conto della reale posizione del punto di partenza.



Costruttore:

```
public Parabola(Point startPoint, Point endPoint) {  
    super(startPoint, endPoint);  
    //calculate a quadratic coefficient  
    a = intervalX / Math.pow(intervalY, 2);  
}
```

Invoca il costruttore della superclasse e, imponendo il passaggio per il punto di arrivo (X_f, Y_f) , ($X_f = \text{intervalX}$ e $Y_f = \text{intervalY}$) calcola a come:

$$X_f = a \cdot Y_f^2 \rightarrow a = \frac{X_f}{Y_f^2}.$$

Metodo che realizza la funzione della parabola:

```
public double evaluateX(double y) {  
    return a * Math.pow(y, 2);  
}
```

Calcola la coordinata x di un punto della parabola in funzione della coordinata y , secondo l'equazione della parabola simmetrica rispetto all'asse x : $x = ay^2$.

Implementazione metodo che restituisce l'array di punti con cui rappresentare la parabola:

```
public Point[] calculatePoints(){  
    //declaration of an array of points of dimension NUMPOINTS  
    Point[] points = new Point[NUMPOINTS];  
    double x;  
    double y;  
    double t;  
    double yCubic;  
  
    for (int i = 0; i < NUMPOINTS; i++) {  
        //t is the index i normalized respect to NUMPOINTS  
        t = (double) i / (NUMPOINTS - 1);  
        //cubic distribution of points along intervalY  
        yCubic = intervalY * Math.pow(t, 3);  
        //addition of startPoint y component  
        y = startPoint.getY() + yCubic;  
        //x coordinate evaluation and addition of startPoint x component  
        x = startPoint.getX() + evaluateX(yCubic);  
        //add the calculated point to the array  
        points[i] = new Point(x, y);  
    }  
    return points;  
}
```

L'errore nell'approssimazione numerica dell'integrale per il calcolo del tempo di discesa è maggiore in prossimità del punto di partenza. Questo avviene perché, in quella regione, la velocità del grave è molto piccola e la funzione integranda essendo il reciproco della velocità, assume dunque valori molto elevati: $T = \int_{y_i}^{y_f} \frac{1}{v_Y} dy$.

Per migliorare la precisione della discretizzazione numerica, è quindi preferibile aumentare la densità dei punti in prossimità del punto di partenza e ridurla progressivamente man mano che si avvicina al punto di arrivo, dove la velocità è maggiore e la discretizzazione può essere meno fine senza compromettere l'accuratezza dell'integrale. Per ottenere questa distribuzione non uniforme, i punti vengono distribuiti lungo l'intervallo di definizione della variabile indipendente y , seguendo una relazione cubica:

- Normalizzazione dell'indice

Si definisce un parametro normalizzato t , che varia da 0 a 1 lungo l'iterazione del ciclo, garantendo una mappatura dell'indice i nell'intervallo $[0,1]$, indipendentemente dal numero di punti.

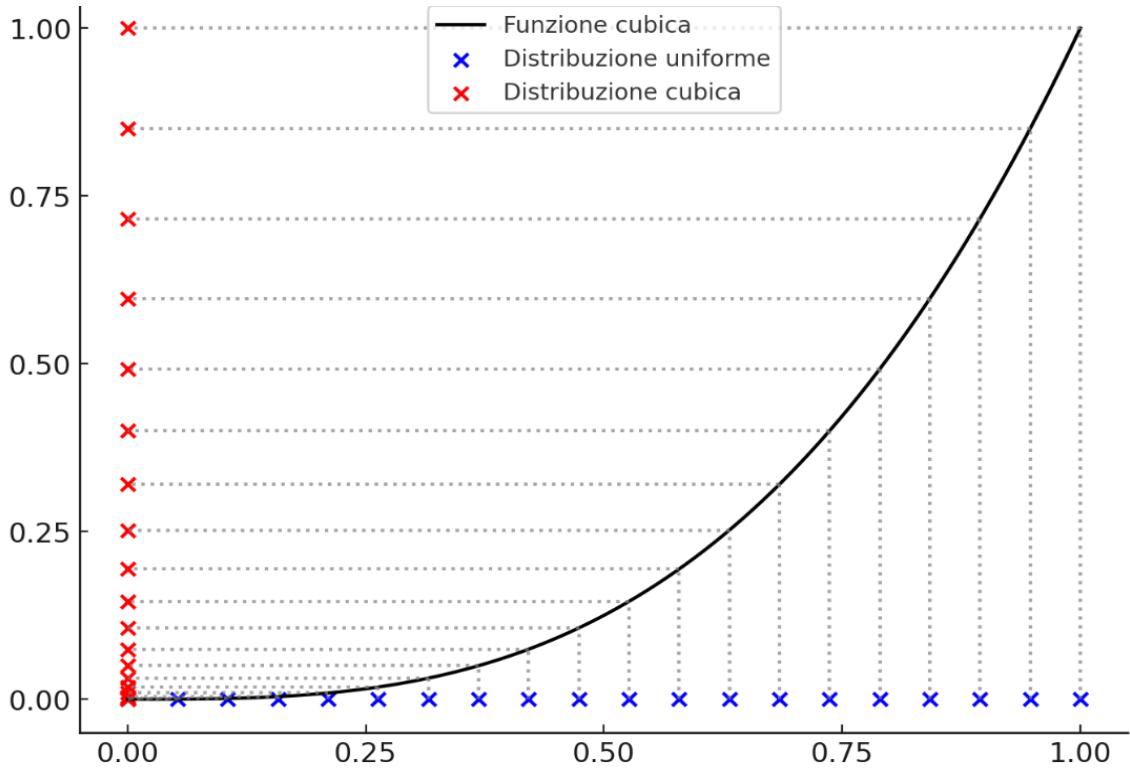
$$t = \frac{i}{NUMPOINTS - 1}.$$

- Trasformazione cubica per la distribuzione non uniforme

Per ottenere una maggiore densità di punti vicino al punto di partenza, si applica una trasformazione cubica:

$$yCubic = intervalY \cdot t^3.$$

Questa funzione fa sì che per valori piccoli di t , la crescita di $yCubic$ sia molto lenta, concentrando così più punti all'inizio. Viceversa, per valori grandi di t , la crescita accelera, portando a una distribuzione più diradata vicino al punto di arrivo.



L'esponente 3 per generare la distribuzione è stato scelto in modo euristico.

intervalY è il fattore per scalare la distribuzione sull'intervallo di definizione y .

- Calcolo delle coordinate del punto

La coordinata Y_P viene ottenuta sommando $yCubic$ alla coordinata y del punto di partenza:

$$Y_P = startPoint.getY() + yCubic.$$

La coordinata X_P viene ottenuta valutando la funzione su $yPow$ e sommando la coordinata X_P del punto di partenza :

$$X_P = startPoint.getX() + evaluateX(yCubic).$$

Il punto (X_P, Y_P) viene quindi aggiunto all'array di punti che rappresentano la parabola distribuiti in modo da concentrarsi maggiormente nel punto di partenza, riducendo così l'errore nella stima dell'integrale numerico del tempo di caduta e migliorando l'accuratezza della simulazione.

Implementazione metodo che restituisce l'array delle pendenze nei punti della parabola:

```
public double[] calculateSlopes(){
    //declaration of an array of slopes of dimension NUMPOINTS
    double[] slopes = new double[NUMPOINTS];
    double yCubic;
    double t;
    for (int i = 0; i < NUMPOINTS; i++) {
        //t is the index i normalized respect to NUMPOINTS
        t = (double) i / (NUMPOINTS - 1);
        //cubic distribution of points along intervalY
        yCubic = intervalY * Math.pow(t, 3);
        //parabola slopes
        slopes[i] = Math.PI / 2 - Math.atan(2 * a * yCubic);
    }
    return slopes;
}
```

Si impiega ancora la stessa distribuzione cubica precedente in modo che le derivate corrispondano correttamente ai punti calcolati con `calculatePoints()`. L'associazione tra i punti della parabola e le relative derivate è garantita dalla corrispondenza degli indici di posizione nei rispettivi array. Poiché entrambi gli array sono generati con la stessa distribuzione, la derivata calcolata in una certa posizione nell'array `slopes` corrisponde esattamente al punto della parabola con lo stesso indice nell'array `points`.

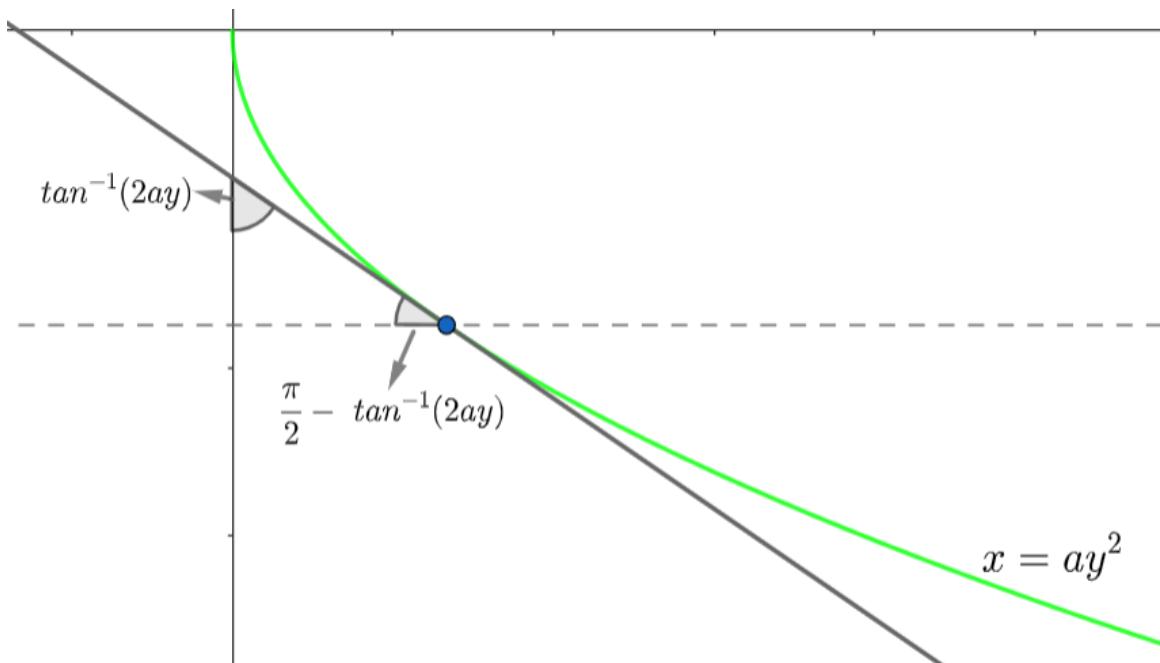
$$\text{points}[i] \rightarrow \text{slopes}[i].$$

Le derivate sono espresse come angoli in radianti di pendenza della curva rispetto all'orizzontale, e si calcolano come:

$$x = ay^2 \rightarrow \frac{dx}{dy} = \frac{d}{dy}(ay^2) = 2ay \rightarrow \frac{dx}{dy} = \frac{1}{2ay}.$$

In radianti:

$$\text{slopes}[i] = \tan^{-1}\left(\frac{1}{2a \cdot yCubic}\right) = \frac{\pi}{2} - \tan^{-1}(2a \cdot yCubic).$$

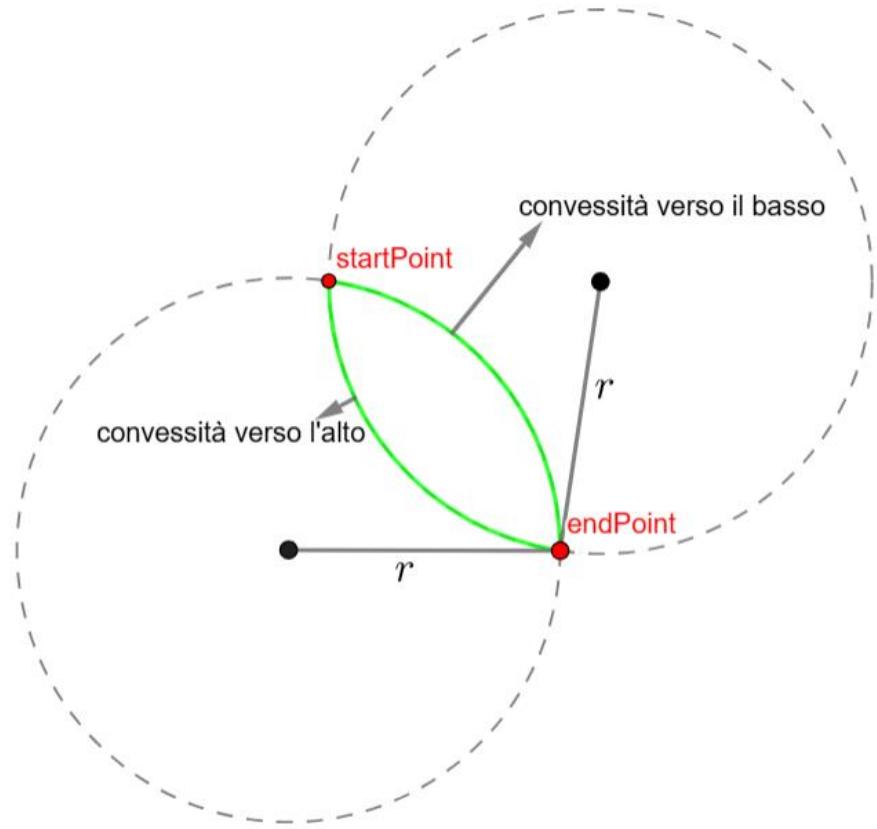


8.7.3. Circonferenza

Definizione classe `Circle` e dichiarazione dei campi:

```
public class Circle extends Curve{
    private double r; //circle radius
    private int convexity; //circle convexity : 1->convexity up
    -1->convexity down
```

Si vuole rappresentare un arco di circonferenza che passi per il punto di partenza e il punto di arrivo. Per definire univocamente questa curva, è necessario specificarne il raggio e la convessità. Quest'ultima è rappresentata da un valore intero: 1 indica una convessità rivolta verso l'alto, mentre -1 indica una convessità rivolta verso il basso.



Si definiscono due costruttori con diverse liste di parametri, secondo la tecnica della programmazione ad oggetti dell'overloading.

Costruttore della circonferenza iniziale:

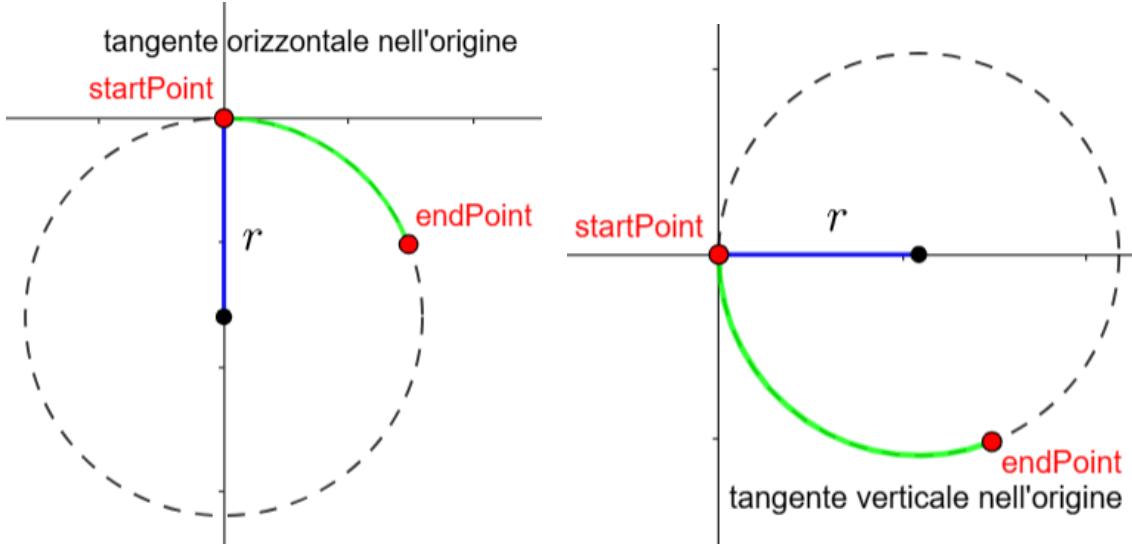
```
//initial circle constructor
public Circumference(Point startPoint, Point endPoint, int convexity){
    super(startPoint, endPoint);
    this.convexity=convexity;
    //convexity down
    if(convexity == -1){
        //radius for horizontal tangent in start point
        this.r = (Math.pow(intervalX, 2)+Math.pow(intervalY, 2))/(2 *
        intervalY) + 1;
    }
    //convexity up
    //radius for vertical tangent in start point
    else this.r = (Math.pow(intervalX, 2)+Math.pow(intervalY, 2))/(2 *
    intervalX);
}
```

Il primo costruttore genera la circonferenza iniziale prima che l'utente abbia ancora selezionato il raggio. Si è stabilito che le circonferenze iniziali siano tali da avere nel punto di partenza, tangente verticale per la circonferenza con convessità verso l'alto e tangente orizzontale per la circonferenza con convessità verso il basso.

Considerando il punto di partenza nell'origine, avere tangente verticale nell'origine significa avere il centro sull'asse x in $(r, 0)$, mentre avere tangente orizzontale nell'origine significa avere il centro sull'asse y in $(0, r)$. Le equazioni delle circonferenze quindi sono:

$$(x - r)^2 + y^2 = r^2 \rightarrow x^2 + y^2 - 2rx = 0 \text{ per convessità verso l'alto.}$$

$$x^2 + (y - r)^2 = r^2 \rightarrow x^2 + y^2 - 2ry = 0 \text{ per convessità verso il basso.}$$



Imponendo il passaggio per il punto di arrivo di coordinate (X_f, Y_f) , ($X_f = intervalX$ e $Y_f = intervalY$) si può ottenere il raggio:

$$r = (X_f^2 + Y_f^2)/(2 \cdot X_f) \text{ per convessità verso l'alto.}$$

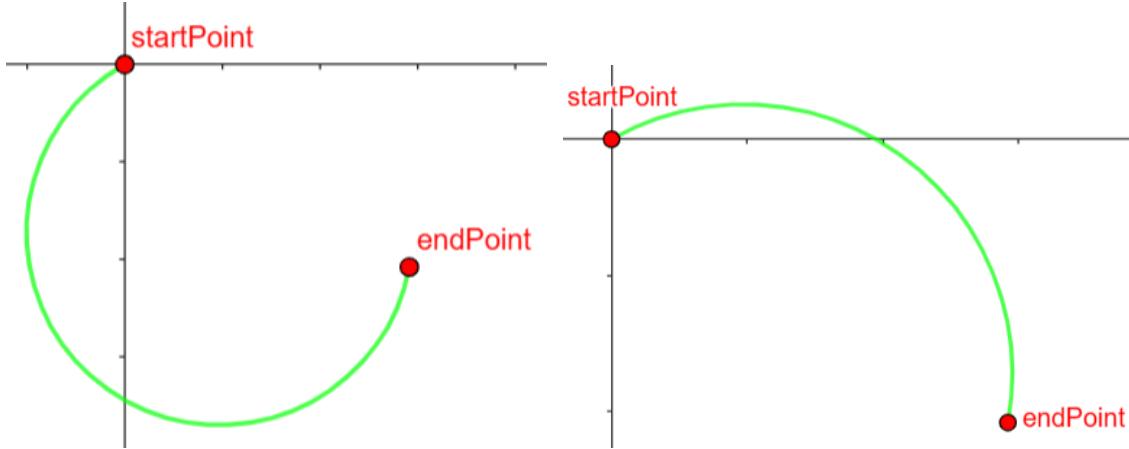
$$r = (X_f^2 + Y_f^2)/(2 \cdot Y_f) \text{ per convessità verso il basso.}$$

Nel caso della circonferenza con convessità verso il basso si somma 1 (un' unità nel sistema di riferimento di JavaFX corrisponde a un solo pixel) alla formula del raggio, per fare in modo di avere almeno un minimo di pendenza nel punto di partenza. Se la tangente fosse esattamente orizzontale la pendenza sarebbe nulla e la massa rimarrebbe in equilibrio nel punto di partenza e non inizierebbe nel suo moto di caduta verso il punto di arrivo.

Costruttore della circonferenza dato il raggio selezionato:

```
//selected radius circle constructor
public Circumference(Point startPoint, Point endPoint, int convexity, double r){
    super(startPoint, endPoint);
    this.r = r;
    this.convexity = convexity;
}
```

Il secondo costruttore genera la circonferenza con il raggio selezionato dall'utente che viene passato come parametro. Si fa in modo che l'utente possa selezionare il raggio r nell'intervallo di valori $[r_{min}, 3r_{min}]$, dove r_{min} è il raggio calcolato dal primo costruttore. Valori minori di r_{min} danno luogo a curve non valide:



Per la circonferenza con convessità verso l'alto si avrebbe che la curva nel punto di partenza inizia nella direzione opposta rispetto al punto di arrivo.

Per la circonferenza con convessità verso il basso si avrebbe che la curva inizia salendo al di sopra del punto di partenza.

Tali situazioni oltre che non essere fisicamente significative per il fenomeno, non sono esprimibili come funzioni. Si vogliono avere invece archi di semicirconferenze divise dagli assi verticale e orizzontale, di modo che si possa calcolare la variabile dipendente in funzione della variabile indipendente. Se la variabile indipendente sia x o y dipende dalla convessità:

Valori maggiori di r_{min} invece sono accettabili perché corrispondono ad archi di semicirconferenze esprimibili come funzioni. Si stabilisce il limite superiore a $3r_{min}$ perché per tale valore del raggio, l'arco di circonferenza diventa simile al collegamento lineare.

Dato il raggio della circonferenza, si calcolano le coordinate relative (X_C, Y_C) del centro rispetto al punto di partenza, che viene in questa fase considerato come l'origine, per semplificare i calcoli. Quindi, imponendo i passaggi per il punto di partenza $(0,0)$ e arrivo (X_f, Y_f) , ($X_f = intervalX$ e $Y_f = intervalY$) si ottiene un sistema di due equazioni nelle incognite X_C , Y_C :

$$\begin{cases} (0 - X_C)^2 + (0 - Y_C)^2 = r^2 \\ (X_f - X_C)^2 + (Y_f - Y_C)^2 = r^2 \end{cases}$$

$$\begin{cases} X_C^2 + Y_C^2 = r^2 \\ (X_f - X_C)^2 + (Y_f - Y_C)^2 = r^2 \end{cases}.$$

Sottraendo la prima dalla seconda:

$$(X_f - X_C)^2 + (Y_f - Y_C)^2 - (X_C^2 + Y_C^2) = 0$$

$$X_f^2 - 2X_f X_C + X_C^2 + Y_f^2 - 2Y_f Y_C + Y_C^2 - X_C^2 - Y_C^2 = 0$$

$$X_f^2 - 2X_f X_C + Y_f^2 - 2Y_f Y_C = 0$$

$$X_f X_C + Y_f Y_C = \frac{X_f^2 + Y_f^2}{2}$$

$$Y_C = \frac{\frac{X_f^2 + Y_f^2}{2} - X_C X_f}{Y_f}$$

$$Y_C = \frac{X_f^2 + Y_f^2 - 2X_C X_f}{2Y_f}.$$

Sostituendo nella prima equazione:

$$X_C^2 + \left(\frac{X_f^2 + Y_f^2 - 2X_C X_f}{2Y_f} \right)^2 = r^2$$

$$X_C^2 + \frac{X_f^4 + Y_f^4 + 4X_C^2 X_f^2 + 2X_f^2 Y_f^2 - 4X_C X_f^3 - 4X_C X_f Y_f^2}{4Y_f^2} = r^2$$

$$4X_C^2 Y_f^2 + X_f^4 + Y_f^4 + 4X_C^2 X_f^2 + 2X_f^2 Y_f^2 - 4X_C X_f^3 - 4X_C X_f Y_f^2 = 4Y_f^2 r^2$$

$$4(X_f^2 + Y_f^2) X_C^2 - 4(X_f^3 + X_f Y_f^2) X_C + X_f^4 + Y_f^4 + 2X_f^2 Y_f^2 - 4Y_f^2 r^2 = 0$$

$$4(X_f^2 + Y_f^2) X_C^2 - 4(X_f^3 + X_f Y_f^2) X_C + (X_f^2 + Y_f^2)^2 - 4Y_f^2 r^2 = 0$$

$$(X_f^2 + Y_f^2) X_C^2 - X_f (X_f^2 + Y_f^2) X_C + \left(\frac{X_f^2 + Y_f^2}{2} \right)^2 - Y_f^2 r^2 = 0.$$

Per ottenere la coordinata x del centro bisogna risolvere un'equazione di secondo grado parametrica rispetto a X_f, Y_f e r .

Metodi che calcolano i coefficienti dell'equazione:

```
//second order equation coefficients
private double aCoefficient(){
    return Math.pow(intervalX, 2) + Math.pow(intervalY, 2);
}


$$a = X_f^2 + Y_f^2.$$


private double bCoefficient(){
    return -intervalX*aCoefficient();
}


$$b = -X_f(X_f^2 + Y_f^2) = -X_f a.$$


private double cCoefficient(){
    return Math.pow(aCoefficient()/2, 2) - Math.pow(intervalY*r, 2);
}


$$c = \left(\frac{X_f^2 + Y_f^2}{2}\right)^2 - Y_f^2 r^2 = \left(\frac{a}{2}\right)^2 - (Y_f r)^2.$$

```

Metodo che calcola la coordinata x del centro X_C :

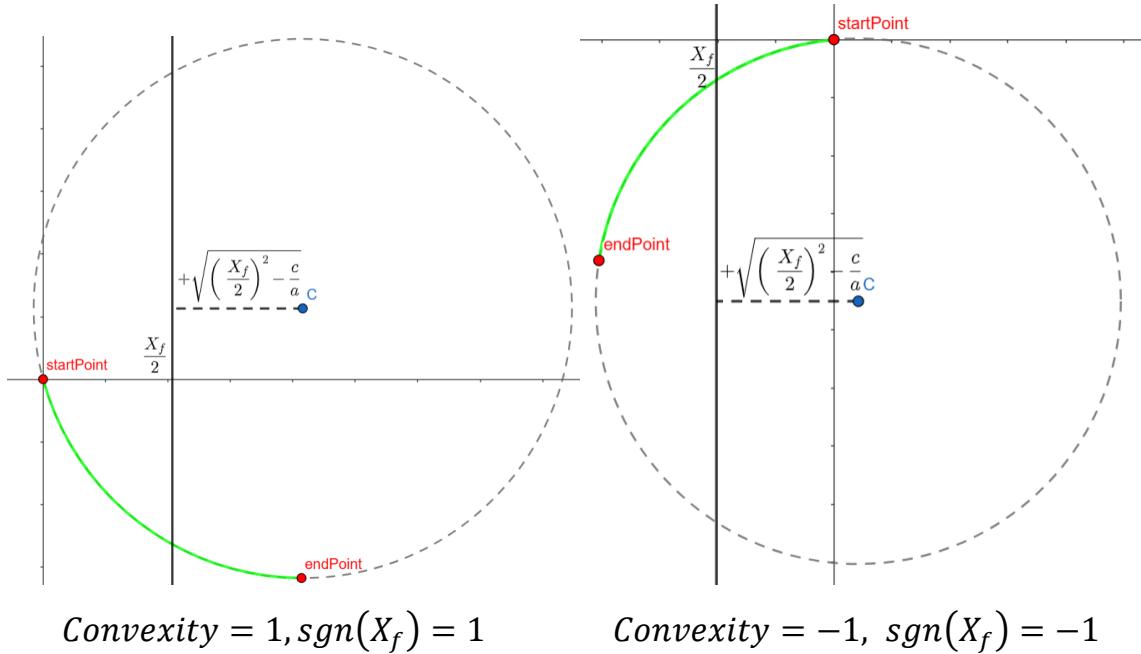
```
//calculates center relative X coordinate
private double xCenter(){
    double xCenter = intervalX/2 + convexity*Math.signum(intervalX) *
        Math.sqrt(Math.pow(intervalX/2, 2) - cCoefficient()/aCoefficient());
    return xCenter;
}


$$X_C = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-(-X_f a)}{2a} \pm \sqrt{\frac{(X_f a)^2 - 4ac}{4a^2}} = \frac{X_f}{2} \pm \sqrt{\left(\frac{X_f}{2}\right)^2 - \frac{c}{a}}.$$

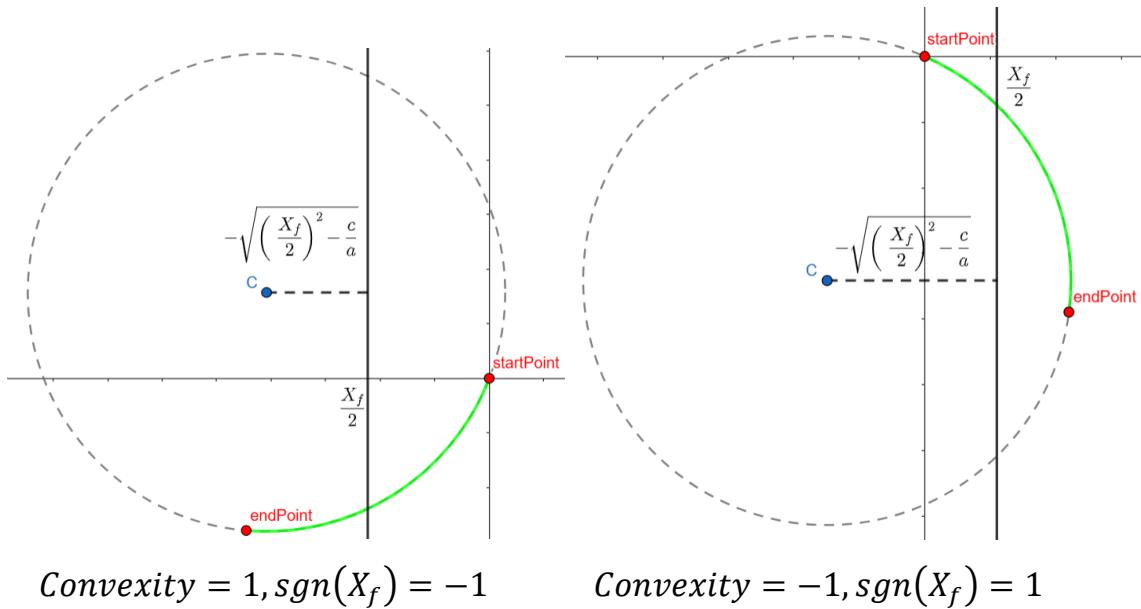
```

Per la scelta del segno del secondo termine bisogna tenere conto sia della convessità che del segno di X_f .

- Convessità e $sgn(X_f)$ concordi: segno + .



- Convessità e $sgn(X_f)$ discordi: segno - .



Quindi il segno del secondo termine di X_C è dato da: $convexity \cdot sgn(X_f)$.

$$X_C = \frac{X_f}{2} + convexity \cdot sgn(X_f) \cdot \sqrt{\left(\frac{X_f}{2}\right)^2 - \frac{c}{a}}.$$

Metodo che calcola la coordinata y del centro Y_C :

```
//calculates center relative Y coordinate
private double yCenter(){
    double yCenter = (Math.pow(intervalX, 2) + Math.pow(intervalY, 2) -
    2*xCenter()*intervalX)/(2*intervalY);
    return yCenter;
}
```

$$Y_C = \frac{X_f^2 + Y_f^2 - 2X_C X_f}{2Y_f}.$$

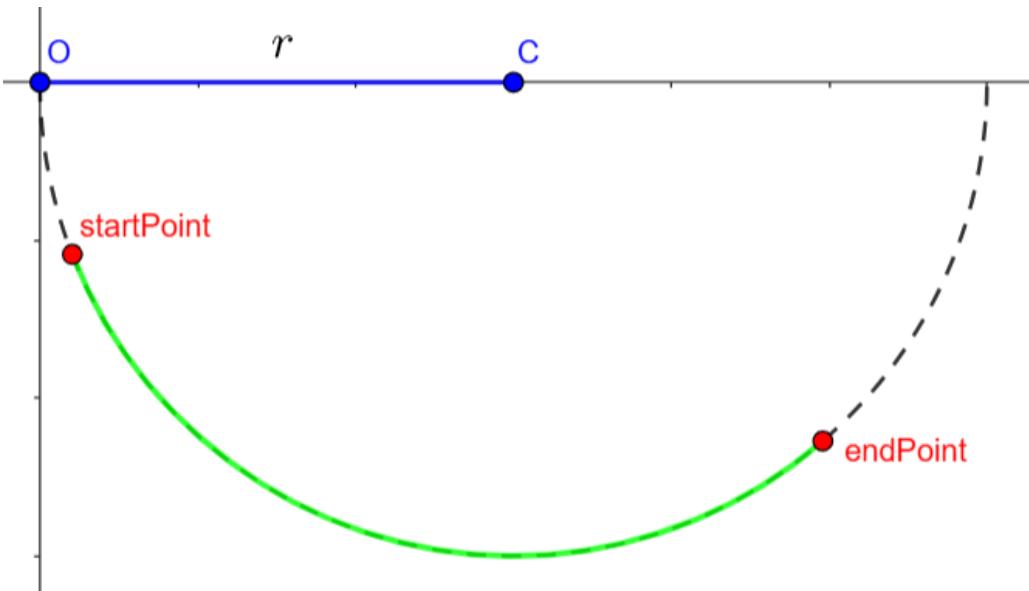
Metodo che realizza la funzione semicirconferenza:

```
//function for a circle passing from the origin
public double evaluateFunction(double variable) {
    return Math.sqrt(2*variable*r - Math.pow(variable, 2));
}
```

Non viene specificata quale è la variabile nell'argomento perché dipende dalla convessità, ma la forma dell'espressione è la stessa:

- Convessità verso l'alto: semicirconferenza di centro $(r, 0)$, superiore rispetto all'asse x .

$$y = \sqrt{2rx - x^2}.$$

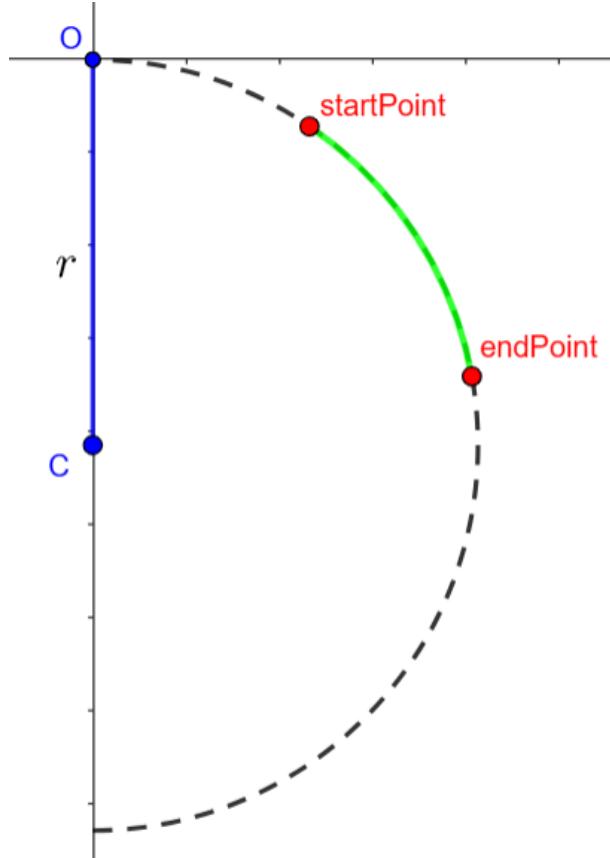


Il punto O a cui riferire le coordinate per la funzione `evaluateFunction` ha coordinate assolute:

$$X_O = (X_C - r, Y_C).$$

- Convessità verso il basso: semicirconferenza di centro $(0, r)$, a destra rispetto all'asse y .

$$x = \sqrt{2ry - y^2}.$$



Il punto O a cui riferire le coordinate per la funzione `evaluateFunction` ha coordinate assolute:

$$O = (X_C, Y_C - r).$$

Implementazione metodo che restituisce l'array di punti con cui rappresentare l'arco di circonferenza:

```
@Override
public Point[] calculatePoints() {
    //declaration of an array of points of dimension NUMPOINTS
    Point[] points = new Point[NUMPOINTS];
    //absolute coordinates of the center
    double xCenter = xCenter() + startPoint.getX();
    double yCenter = yCenter() + startPoint.getY();
    double x;
    double y;
```

```

//convexity up
if(convexity == 1){
    double t;
    double xCubic;
    //origin for evaluateFunction to refer coordinates
    double x0 = xCenter - r;
    for (int i=0; i < NUMPOINTS; i++) {
        //t is the index i normalized respect to NUMPOINTS
        t = (double) i / (NUMPOINTS - 1);
        //cubic distribution of points along intervalX
        xCubic = intervalX * Math.pow(t, 3);
        //addition of startPoint x component
        x = startPoint.getX() + xCubic;
        //y coordinate evaluation
        y = yCenter + evaluateFunction(x - x0);
        //add the calculated point to the array
        points[i] = new Point(x,y);
    }
}
//convexity down
else if(convexity == -1){
    double t;
    double yCubic;
    //origin for evaluateFunction to refer coordinates
    double y0 = yCenter - r;
    for (int i=0; i < NUMPOINTS; i++) {
        //t is the index i normalized respect to NUMPOINTS
        t = (double) i / (NUMPOINTS - 1);
        //cubic distribution of points along intervalY
        yCubic = intervalY * Math.pow(t, 3);
        //addition of startPoint y component
        y = startPoint.getY() + yCubic;
        //x coordinate evaluation
        x = xCenter + Math.signum(intervalX)*evaluateFunction(y - y0);
        //add the calculated point to the array
        points[i] = new Point(x,y);
    }
}
return points;
}

```

Si calcolano le coordinate assolute del centro della circonferenza, sommando alle coordinate relative calcolate con i metodi `xCenter()`, `yCenter()`, le coordinate del punto di partenza.

Come per la parabola viene utilizzata una distribuzione cubica per la variabile indipendente. Per la circonferenza con convessità verso l'alto la variabile indipendente è

x , mentre per la circonferenza con convessità verso il basso la variabile indipendente è y .

La variabile dipendente viene calcolata dalla funzione `evaluateFunction`. Le coordinate restituite da tale funzione sono relative al punto O , pertanto bisogna sottrarre la sua componente alla variabile indipendente per passarla come parametro alla funzione e poi al risultato bisogna sommare l'altra componente di O , per ottenere la variabile dipendente in coordinate assolute:

- Convessità verso l'alto:

Per calcolare le coordinate di un generico punto P dell'arco di circonferenza:

$$X_P = \text{startPoint}.getX() + xCubic .$$

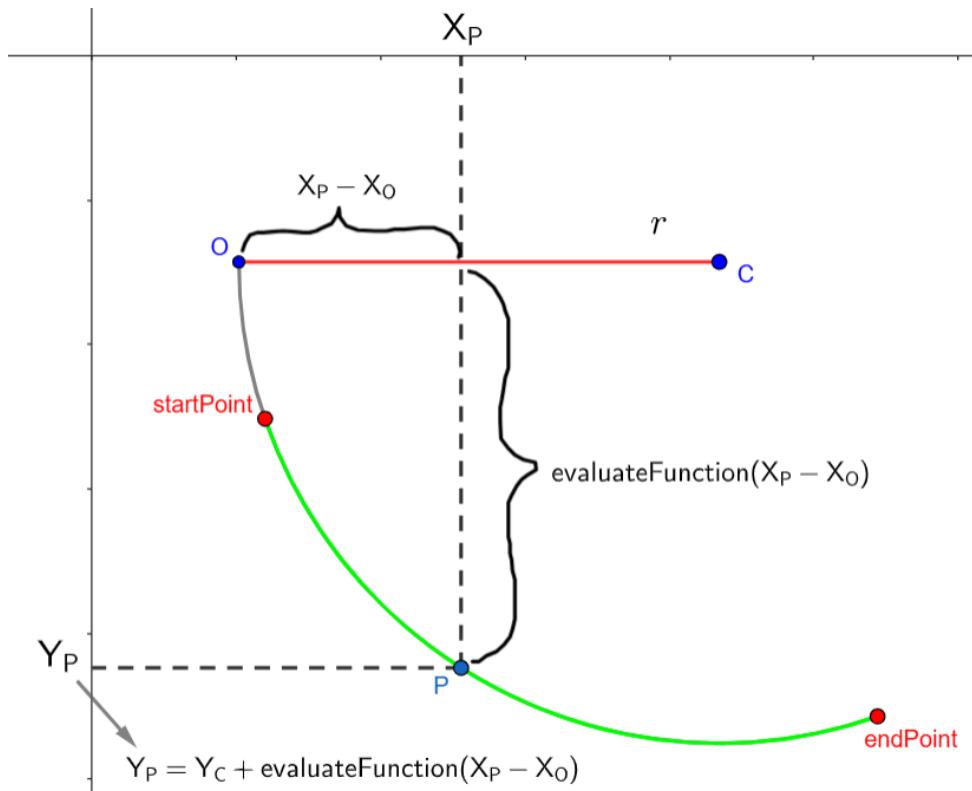
`evaluateFunction` necessita come argomento la coordinata x del punto relativamente al punto $O = (X_C - r, Y_C)$:

$$X_O = X_C - r$$

$$\text{evaluateFunction}(X_P - X_O).$$

La funzione restituisce la coordinata y del punto di arrivo ancora relativamente al punto O . Per ottenere la Y_P assoluta bisogna dunque sommare $Y_O = Y_C$:

$$Y_P = Y_C + \text{evaluateFunction}(X_P - X_O).$$



- Convessità verso il basso:

Analogamente al precedente ma la variabile indipendente è y :

$$Y_P = startPoint.getY() + yCubic$$

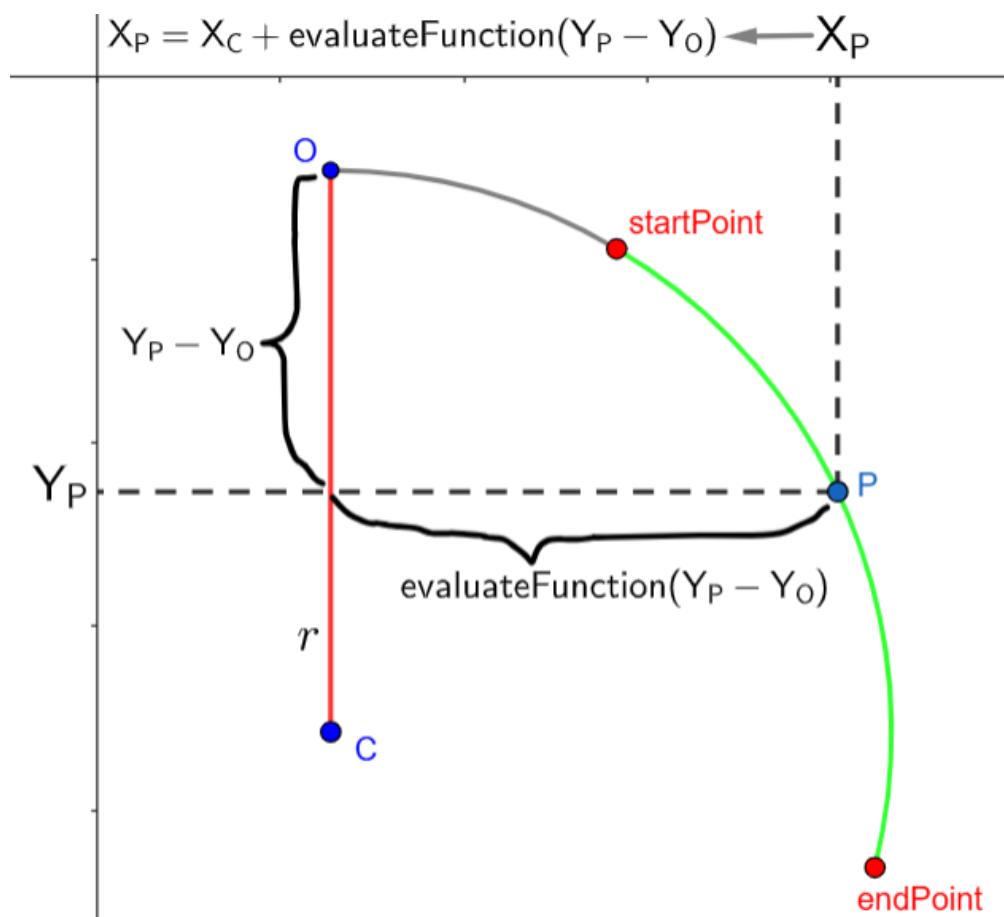
$$Y_O = Y_C - r$$

$$evaluateFunction(X_P - X_O).$$

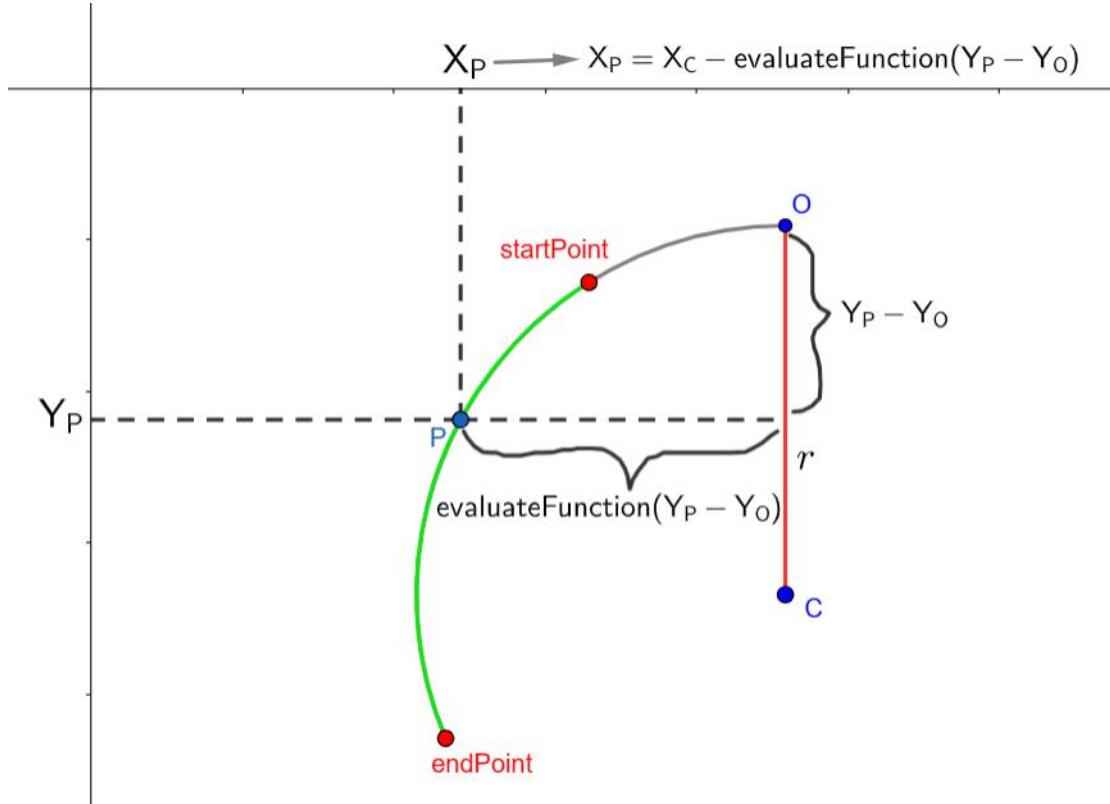
Bisogna tenere conto in questo caso del segno di `intervalX` per sapere se sommare o sottrarre:

$$X_P = X_C + sgn(intervalX) * evaluateFunction(Y_P - Y_O).$$

- $sgn(intervalX) = 1$



- $sgn(intervalX) = -1$



Implementazione metodo che restituisce l'array delle pendenze nei punti dell'arco di circonferenza:

```

@Override
public double[] calculateSlopes(){
    //declaration of an array of slopes of dimension NUMPOINTS
    double[] slopes = new double[NUMPOINTS];

    //absolute coordinates of the center
    double xCenter = xCenter() + startPoint.getX();
    double yCenter = yCenter() + startPoint.getY();

    //convexity up
    if(convexity == 1){
        double t;
        double xCubic;
        double x;
        //origin to refer coordinates
        double x0 = xCenter-r;
        for (int i=0; i < NUMPOINTS; i++) {
            //t is the index i normalized respect to NUMPOINTS
            t = (double) i / (NUMPOINTS - 1);
            //cubic distribution of points along intervalX
            xCubic = intervalX * Math.pow(t, 3);
    
```

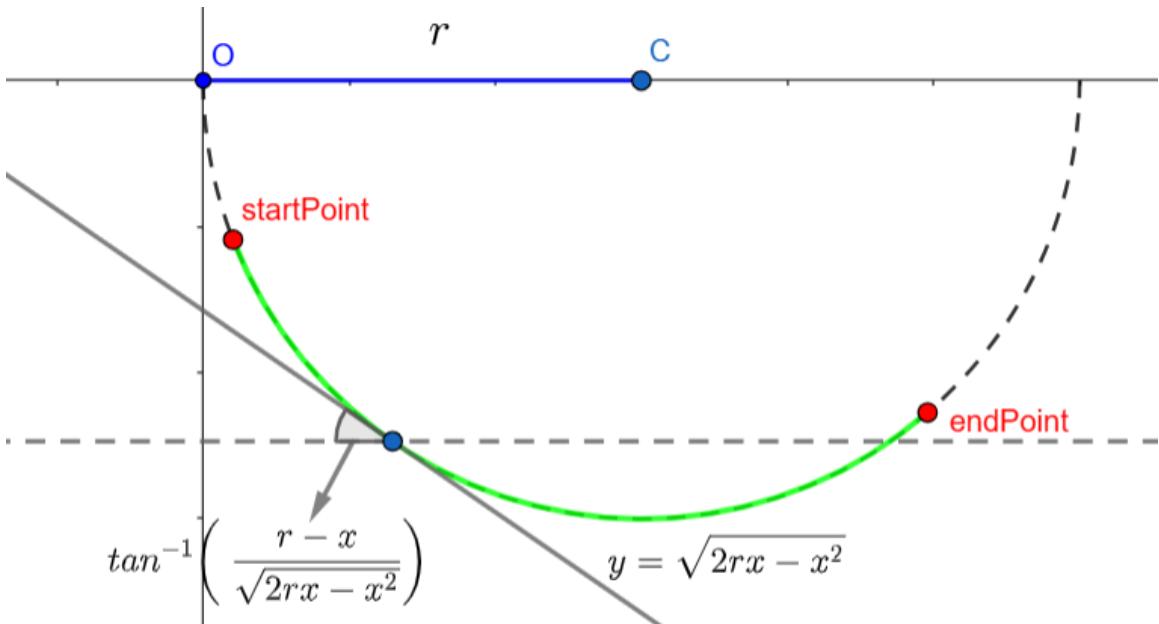
```

        //absolute x coordinate of a generic point
        x = startPoint.getX() + xCubic;
        //circle slope
        slopes[i] = Math.atan((r-(x-x0))/Math.sqrt(2*r*(x-x0)-
        Math.pow(x-x0, 2)));
    }
}
else if(convexity == -1){
    double t;
    double yCubic;
    double y;
    //origin to refer coordinates
    double y0 = yCenter-r;
    for (int i=0; i < NUMPOINTS; i++) {
        //t is the index i normalized respect to NUMPOINTS
        t = (double) i / (NUMPOINTS - 1);
        //cubic distribution of points along intervalX
        yCubic = intervalY * Math.pow(t, 3);
        //absolute y coordinate of a generic point
        y = startPoint.getY() + yCubic;
        //circle slope
        slopes[i] = Math.PI/2 - Math.atan((r-(y-
        y0))/Math.sqrt(2*r*(y-y0) - Math.pow(y-y0, 2)));
    }
}
return slopes;
}

```

- Convessità verso l'alto:

$$y = \sqrt{2rx - x^2} \rightarrow \frac{dy}{dx} = \frac{d}{dx}(\sqrt{2rx - x^2}) = \frac{2r - 2x}{2\sqrt{2rx - x^2}} = \frac{r - x}{\sqrt{2rx - x^2}}.$$

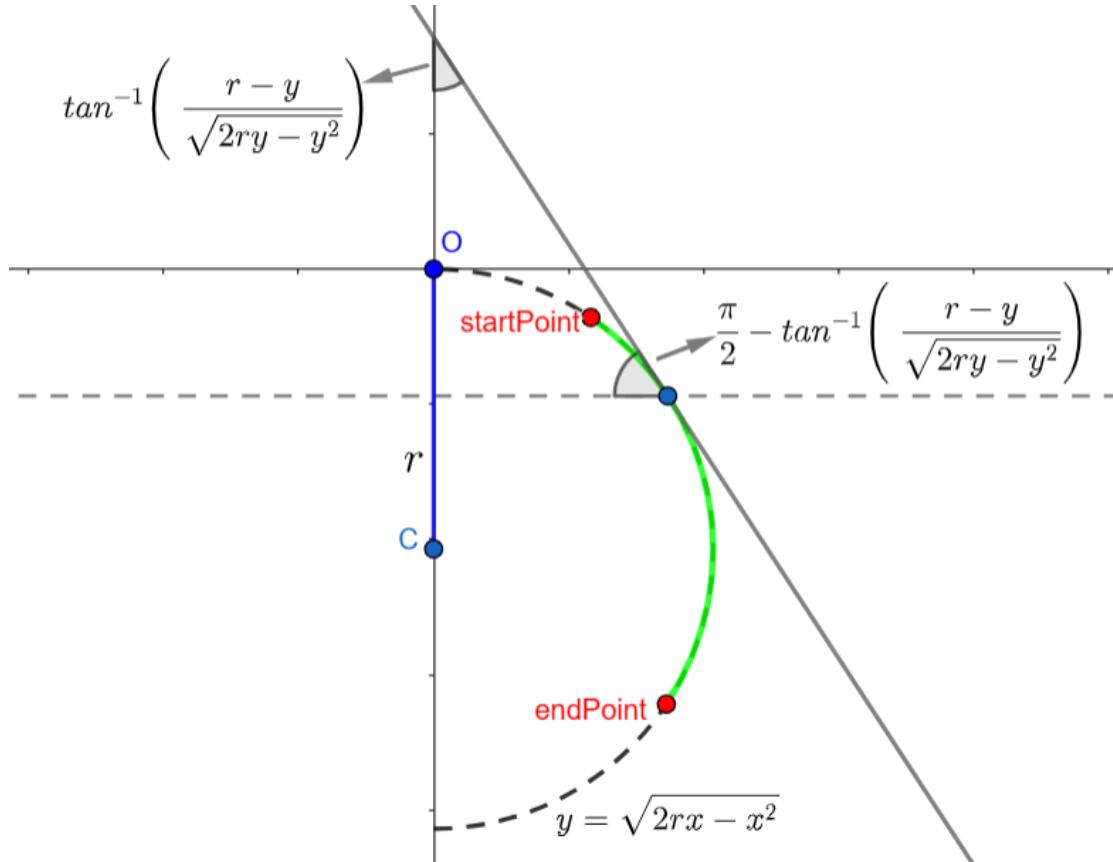


Questa espressione vale per la circonferenza di centro $(r, 0)$, quindi le coordinate devono essere espresse relativamente al punto $O = (X_C - r, Y_C)$. Per usare la coordinata x assoluta del punto bisogna effettuare la traslazione: $x - X_O$:

$$\text{slopes[i]} = \tan^{-1} \left(\frac{r - (x - X_O)}{\sqrt{2r(x - X_O) - (x - X_O)^2}} \right).$$

- convessità verso il basso:

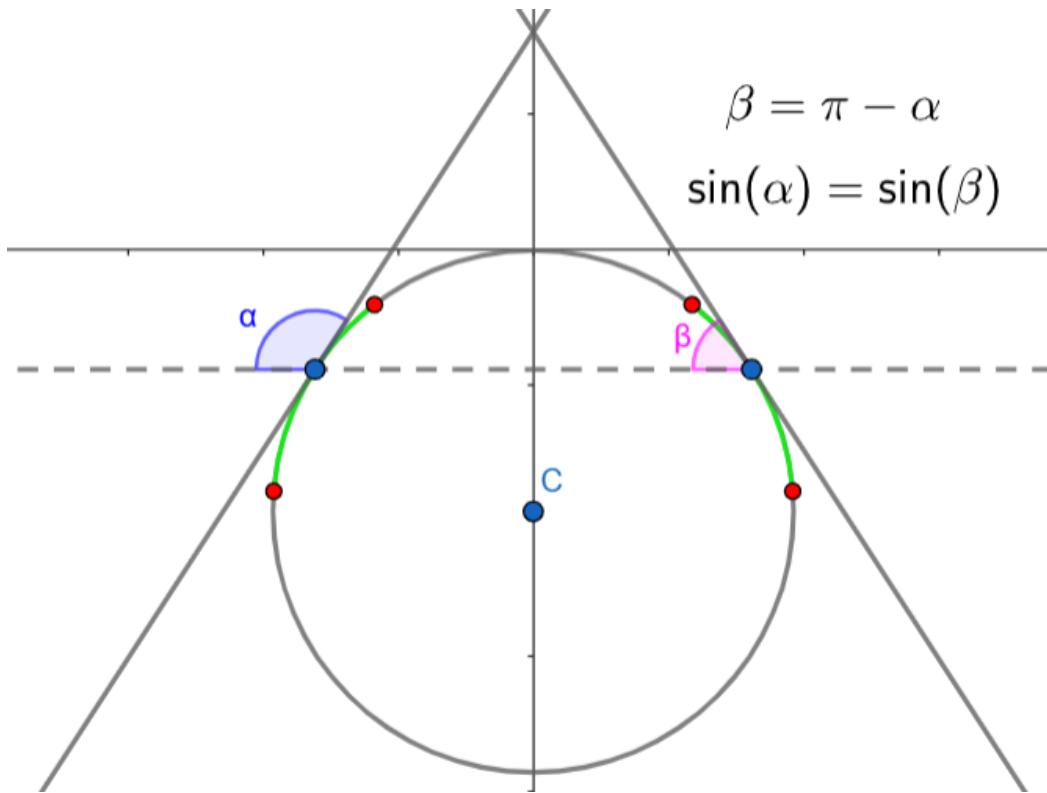
$$x = \sqrt{2ry - y^2} \rightarrow \frac{dx}{dy} = \frac{d}{dy}(\sqrt{2ry - y^2}) = \frac{2r - 2y}{2\sqrt{2ry - y^2}} = \frac{r - y}{\sqrt{2ry - y^2}}.$$



Questa espressione vale per la circonferenza di centro $(0, r)$, quindi le coordinate devono essere espresse relativamente al punto $O = (X_C, Y_C - r)$. Per usare la coordinata y assoluta del punto bisogna effettuare la traslazione $y - Y_O$:

$$\text{slopes[i]} = \frac{\pi}{2} - \tan^{-1} \left(\frac{r - (y - Y_O)}{\sqrt{2r(y - Y_O) - (y - Y_O)^2}} \right).$$

Questa espressione è corretta anche per il caso `intervalX < 0`, poiché per calcolare la componente verticale della velocità nel punto bisogna considerare il seno della pendenza della tangente e quindi non importa se si considera un angolo o il suo supplementare.



8.7.4. Cicloide

Definizione classe `Cycloid` e dichiarazione dei campi:

```
public class Cycloid extends Curve {

    private double alfa; //definition interval for the angular parameter
    private double r; // radius of the generator circle
```

Si vuole rappresentare un arco di cicloide con tangente verticale nel punto di partenza, cioè la curva brachistocrona nella situazione in cui la massa ha velocità nulla nel punto di partenza. Quindi `startPoint` si considera come l'origine del sistema di riferimento in cui descrivere la cicloide.

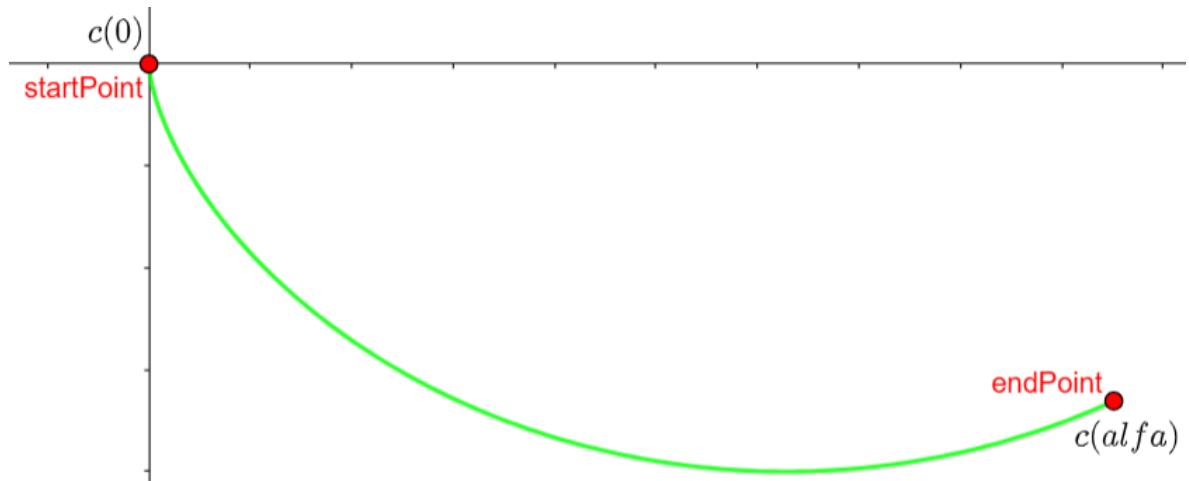
Equazione parametrica della cicloide con tangente verticale nell'origine:

$$c(\alpha) = \begin{cases} x = r \cdot (\alpha - \sin \alpha) \\ y = r \cdot (1 - \cos \alpha) \end{cases}$$

Il campo `alfa` rappresenta l'ampiezza dell'intervallo di definizione angolare $c(\alpha) : [0, \alpha] \rightarrow R^2$ dell'arco di cicloide. Quindi $c(0) = \text{startPoint}$ e $c(\alpha) = \text{endPoint}$.

Poiché si vuole ottenere un arco di cicloide, sottoinsieme di un periodo: $\alpha \leq 2\pi$.

Il campo r rappresenta il raggio della circonferenza generatrice della cicloide.



Costruttore:

```
public     Cycloid(Point      startPoint,      Point      endPoint)      throws
NonConvergenceException {
    super(startPoint, endPoint);
    alfa = calculateAlfa();
    r = calculateR();
}
```

Dati i punti di partenza e arrivo il costruttore inizializza i campi, calcolandoli con appositi metodi di servizio definiti nella classe stessa:

Metodo che calcola il campo α :

```
// Newton-Raphson method for finding alfa
private double calculateAlfa() throws NonConvergenceException {
    // good initial approximation
    double alfaLocal = 4 * Math.atan(intervalX / (2 * intervalY));
    // maximum number of iterations
    int maxIterations = 100;
    // Newton-Raphson iterations
    for (int i = 0; i < maxIterations; i++) {
        //new value for alfa
        double alfaNew = alfaLocal - f(alfaLocal) / df(alfaLocal);
        //convergence control
        if (Math.abs(alfaNew - alfaLocal) < 1e-6) {
            Logger.info("Convergenza raggiunta: alfa = {}", alfaNew);
            // return value for alfa when the approximation is good enough
            return alfaNew;
        }
        //alfa value update for next iteration
        alfaLocal = alfaNew;
    }
}
```

```

Logger.error("Il metodo di Newton-Raphson non converge dopo {} 
iterazioni", maxIterations);
//throw exception when Newton-Raphson method doesn't converge
throw new NonConvergenceException(maxIterations);
}

```

Il metodo `calculateAlfa` riceve come parametri le coordinate relative del punto di arrivo rispetto al punto di partenza.

Imponendo il passaggio per il punto di arrivo (X_f, Y_f) , ($X_f = intervalX$ e $Y_f = intervalY$)

$$c(\alpha) = \text{endPoint}.$$

$$\begin{cases} X_f = r \cdot (\alpha - \sin(\alpha)) \\ Y_f = r \cdot (1 - \cos(\alpha)) \end{cases}.$$

Dividendo la prima equazione per la seconda:

$$\frac{X_f}{Y_f} = \frac{(\alpha - \sin(\alpha))}{(1 - \cos(\alpha))} \rightarrow \frac{(\alpha - \sin(\alpha))}{(1 - \cos(\alpha))} - \frac{X_f}{Y_f} = 0.$$

Dal momento che questa equazione non è risolvibile analiticamente, α viene calcolato numericamente tramite il metodo di Newton-Raphson. Questo procedimento iterativo parte da una stima iniziale e aggiorna il valore ad ogni iterazione fino a raggiungere una precisione definita.

Metodo che calcola la funzione per cui si vogliono trovare gli zeri con il metodo di Newton-Raphson:

```

//function to calculate zeros with Newton-Raphson method
private double f(double a) {
    return ((a - Math.sin(a)) / (1 - Math.cos(a))) - (intervalX / intervalY);
}

```

$$f(\alpha) = \frac{(\alpha - \sin \alpha)}{(1 - \cos \alpha)} - \frac{X_f}{Y_f}.$$

Metodo che calcola la derivata della funzione $f(\alpha)$:

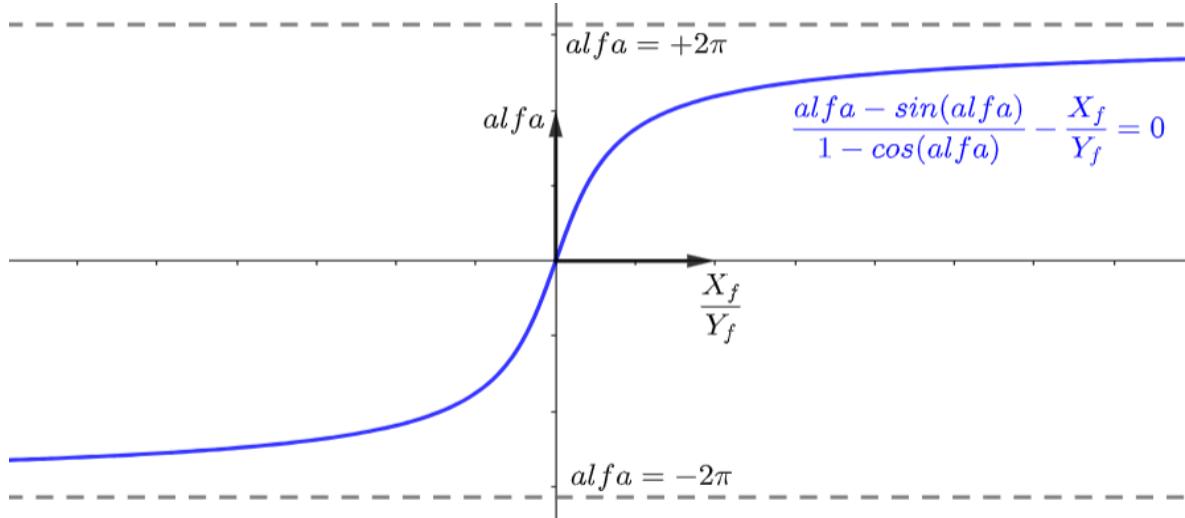
```

//derivative of the function
private double df(double a) {
    double numerator = Math.pow(Math.sin(a), 2) - a * Math.sin(a);
    double denominator = Math.pow(1 - Math.cos(a), 2);
    return 1 + numerator / denominator;
}

```

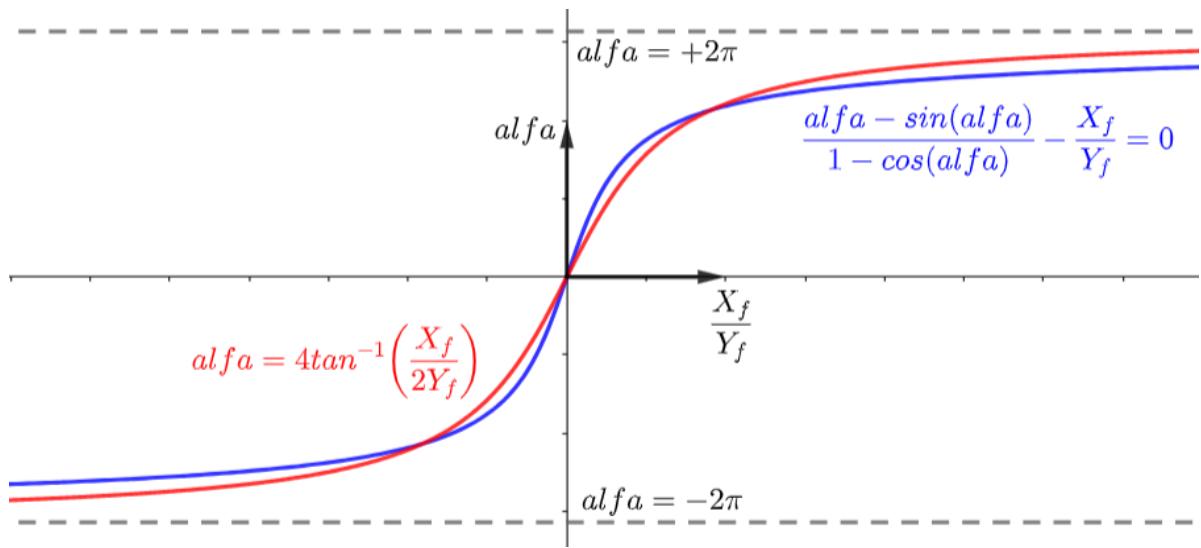
$$\frac{df}{d\alpha} = \frac{(1 - \cos \alpha)^2 + \sin \alpha \cdot (\alpha - \sin \alpha)}{(1 - \cos \alpha)^2} = 1 + \frac{\sin \alpha \cdot (\alpha - \sin \alpha)}{(1 - \cos \alpha)^2}.$$

Graficando lo zero α di $f(\alpha)$ in funzione di $\frac{X_f}{Y_f}$:



Una buona stima iniziale di α , per il metodo di Newton-Raphson, è data da:

$$\alpha_0 = 4 \tan^{-1} \left(\frac{X_f}{2Y_f} \right).$$



Avere una buona stima iniziale, permette di convergere rapidamente alla soluzione dell'equazione, migliorando così le prestazioni.

Si definisce un limite massimo al numero di iterazioni `maxIterations = 100`, superate le quali si segnala la mancata convergenza del metodo e viene lanciata un'eccezione `NonConvergenceException(maxIterations)`, definita appositamente.

Eccezione in caso di mancata convergenza del metodo di Newton-Raphson:

```
public class NonConvergenceException extends Exception {
    //exception defined for non convergence of Raphson Newton method for the
    cycloid
    public NonConvergenceException(int numberIterations) {
        super("Il metodo non converge dopo " + numberIterations + " "
              + "iterazioni.");
    }
}
```

Ad ogni iterazione viene calcolata un'approssimazione migliore per `alfa` :

$$\alpha_{n+1} = \alpha_n - \frac{f(\alpha_n)}{f'(\alpha_n)}.$$

Metodo che calcola il campo `r`:

```
private double calculateR() {
    double r = intervalY / (1 - Math.cos(alfa));
    return r;
}
```

Una volta calcolato il parametro `alfa` è possibile calcolare il raggio semplicemente come:

$$Y_f = r \cdot (1 - \cos(\alpha)) \rightarrow r = \frac{Y_f}{1 - \cos(\alpha)}.$$

Metodi che realizzano la funzione parametrica della cicloide, calcolando le coordinate x e y di un punto in funzione del parametro angolare α :

```
//x component parametric function for a cycloid with vertical tangent in the
origin
public double evaluateX(double a) {
    return r * (a - Math.sin(a));
}
```

$$x = r \cdot (\alpha - \sin \alpha).$$

```
//y component parametric function for a cycloid with vertical tangent in the
origin
public double evaluateY(double a) {
    return r * (1 - Math.cos(a));
}
```

$$y = r \cdot (1 - \cos \alpha).$$

Implementazione metodo che restituisce l'array di punti con cui rappresentare l'arco di cicloide:

```
public Point[] calculatePoints() {
    //declaration of an array of points of dimension NUMPOINTS
    Point[] points = new Point[NUMPOINTS];
    double t;
    double aCubic;
    double x;
    double y;
    for (int i = 0; i < NUMPOINTS; i++){
        //t is the index i normalized respect to NUMPOINTS
        t = (double) i / (NUMPOINTS - 1);
        //cubic distribution of points along [0,alfa]
        aCubic = alfa * Math.pow(t, 3);
        //x coordinate evaluation and addition of startPoint x component
        x = startPoint.getX() + evaluateX(aCubic);
        //y coordinate evaluation and addition of startPoint y component
        y = startPoint.getY() + evaluateY(aCubic);
        //add the calculated point to the array
        points[i] = new Point(x, y);
    }
    return points;
}
```

Per la cicloide la variabile indipendente su cui viene eseguita la distribuzione cubica è il parametro angolare α , su cui viene eseguita la distribuzione cubica nell'intervallo $[0, \alpha]$.

$$aCubic = alfa \cdot t^3.$$

Le coordinate X_P e Y_P dei punti della cicloide vengono calcolate come variabili dipendenti:

$$X_P = startPoint.getX() + evaluateX(aCubic).$$

$$Y_P = startPoint.getY() + evaluateY(aCubic).$$

Implementazione metodo che restituisce l'array delle pendenze nei punti dell'arco di cicloide:

```
public double[] calculateSlopes() {
    //declaration of an array of slopes of dimension NUMPOINTS
    double[] slopes = new double[NUMPOINTS];
    double t;
    double aCubic;
    //vertical tangent in startPoint
    slopes[0] = Math.PI / 2;
    for (int i = 1; i < NUMPOINTS; i++) {
```

```

    //t is the index i normalized respect to NUMPOINTS
    t = (double) i / (NUMPOINTS - 1);
    //cubic distribution of points along [0,alfa]
    aCubic = alfa * Math.pow(t, 3);
    //cycloid slope
    slopes[i] = Math.atan(Math.sin(aCubic) / (1 - Math.cos(aCubic)));
}
return slopes;
}

```

Si calcolano le derivate prime rispetto al parametro per calcolare poi la pendenza della tangente alla curva:

$$\frac{dx}{d\alpha} = \frac{d}{d\alpha}(r(\alpha - \sin \alpha)) = r(1 - \cos \alpha)$$

$$\frac{dy}{d\alpha} = \frac{d}{d\alpha}(r(1 - \cos \alpha)) = r \cdot \sin \alpha.$$

Per $\alpha = 0$, cioè in `startPoint`, la tangente è verticale. Per questo si assegna:

$$slopes[0] = \frac{\pi}{2};$$

$\frac{dx}{d\alpha} > 0$, per $0 < \alpha < 2\pi$, quindi $x(\alpha)$ monotona crescente, quindi $x(\alpha)$ invertibile:

$$\frac{dy}{dx} = \frac{dy}{d\alpha} \cdot \frac{d\alpha}{dx} = \frac{r \cdot \sin \alpha}{r \cdot (1 - \cos \alpha)} = \frac{\sin \alpha}{1 - \cos \alpha}.$$

$$slopes[i] = \tan^{-1}\left(\frac{\sin \alpha}{1 - \cos \alpha}\right).$$

8.7.5. Spline cubica

Definizione classe `CubicSpline` e dichiarazione dei campi:

```

public class CubicSpline extends Curve {
    //apache commons math spline function
    private PolynomialSplineFunction splineFunction;
    //array of the points to interpolate
    private Point[] interpolationPoints;
}

```

Si vuole rappresentare una curva per interpolare una sequenza di punti inseriti dall'utente.

A tale scopo si utilizza il campo `splineFunction` della classe `PolynomialSplineFunction`, fornita dalla libreria Apache Commons Math. I punti da interpolare sono memorizzati in un array di punti `interpolationPoints`.



Costruttore:

```
public CubicSpline(Point startPoint, Point endPoint, List<Point> intermediatePoints) {
    super(startPoint, endPoint);
    //dimension of interpolationPoints: #intermediate points + 2
    int n = intermediatePoints.size() + 2;
    interpolationPoints = new Point[n];
    //add startPoint as first point of interpolationPoints
    interpolationPoints[0] = startPoint;
    //add endPoint as of interpolationPoints
    for(int i = 1; i < n-1; i++){
        interpolationPoints[i] = intermediatePoints.get(i-1);
    }
    //add endPoint as last point of interpolationPoints
    interpolationPoints[n-1] = endPoint;
    //order the points respect to x coordinate
    Arrays.sort(interpolationPoints,
    Comparator.comparingDouble(Point::getX));
    //array of x coordinates
    double[] x = new double[n];
    //array of y coordinates
    double[] y = new double[n];

    for(int i=0; i < n; i++)
    {
        x[i] = interpolationPoints[i].getX();
        y[i] = interpolationPoints[i].getY();
    }
```

```

//creates cubic spline interpolating function with Apache Commons Math
if(interpolationPoints.length > 2) {
    SplineInterpolator interpolator = new SplineInterpolator();
    splineFunction = interpolator.interpolate(x, y);
}
else{//if there are no intermediate points splineFunction is null
    splineFunction = null;
}
}

```

Il costruttore riceve come parametri il punto di partenza, il punto di arrivo e una lista di punti intermedi da interpolare, raccolti dalla classe `InputController` con il metodo `addIntermediatePoint(Point p)`, che verifica anche che tutti i punti siano compresi, lungo x , tra il punto di partenza e il punto di arrivo:

```

endPoint.getX() <= p.getX() && startPoint.getX() >= p.getX() ||
endPoint.getX() >= p.getX() && startPoint.getX() <= p.getX().

```

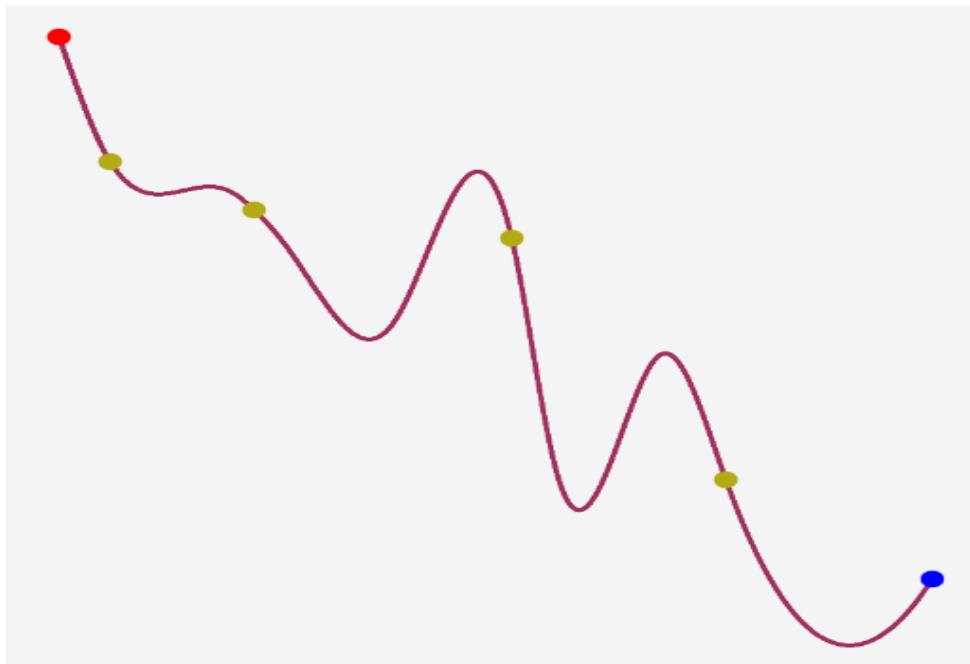
Il campo `interpolationPoints` viene definito come un array di lunghezza la dimensione della lista di punti intermedi più il punto di partenza e il punto di arrivo.

Il punto di partenza viene aggiunto come primo elemento, i punti intermedi inseriti dall'utente vengono aggiunti nel mezzo e il punto di arrivo viene aggiunto come ultimo elemento. Completato il popolamento dell'array, i punti vengono ordinati in ordine crescente rispetto alla coordinata x .

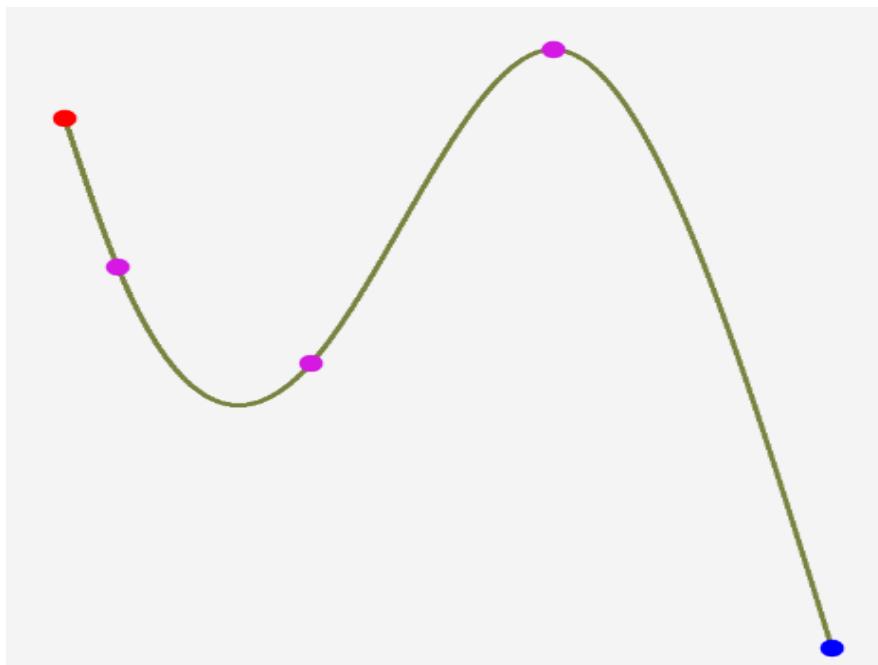
Vengono definiti due nuovi array `x` e `y` della stessa dimensione di `interpolationPoints` contenenti separatamente le coordinate x e y dei punti da interpolare.

A questo punto se è stato inserito almeno un punto intermedio, ovvero se `interpolationPoints.length > 2`, viene creata la spline che interpola i punti, con cui si inizializza il campo `splineFunction`. La spline viene calcolata con il metodo `interpolate`, fornito dalla classe `SplineInterpolator` di Apache Commons Math.

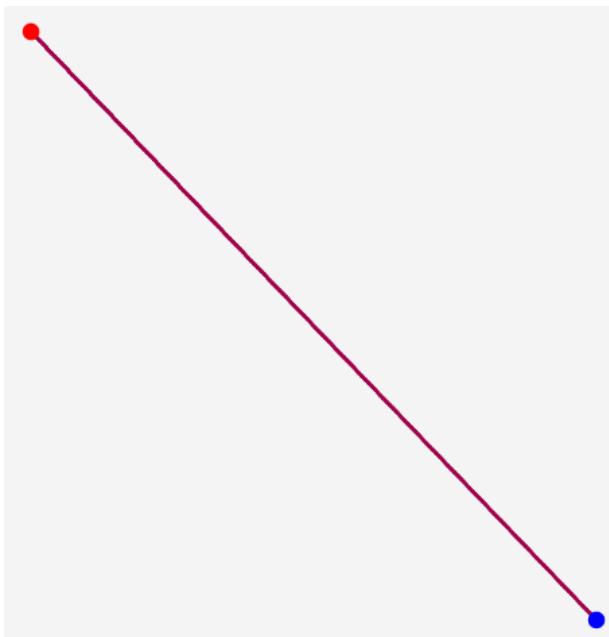
Al metodo vengono passati i due array `x` e `y` come parametri, per effettuare l'interpolazione.



È previsto che sia possibile che la spline raggiunga quote superiori a quella del punto di partenza: in tal caso la massa non riuscirà a raggiungere il punto di arrivo.



Se invece non è stato inserito nessun punto intermedio si inizializza il campo `splineFunction` a `null`, perchè in tal caso si vuole collegare il punto di partenza e quello di arrivo con un segmento lineare.



Metodo che calcola la coordinata y in funzione della coordinata x dei punti con cui si rappresenta la curva:

```
// method for evaluating spline y on a point x
public double evaluateY(double x) {
    //linear segment between startPoint and endPoint
    if(splineFunction == null)
    {
        //angular coefficient
        double m = intervalY / intervalX ;
        //equation of the linear function
        return m * (x - startPoint.getX()) + startPoint.getY();
    }
    //spline function evaluation
    else return splineFunction.value(x);
}
```

Se `splineFunction` è `null` bisogna usare la funzione della retta passante per i punti di partenza e arrivo. Il coefficiente angolare m di tale retta è dato da:

$$m = \frac{endPoint.getY() - startPoint.getY()}{endPoint.getX() - startPoint.getX()} = \frac{intervalX}{intervalY}.$$

Quindi l'equazione della retta è:

$$y = m(x - startPoint.getX()) + startPoint.getY() .$$

Altrimenti se `splineFunction` non è `null`, la coordinata y è restituita dal metodo `value` di `PolynomialSplineFunction`, che riceve come parametro la coordinata x .

Implementazione metodo che restituisce l'array di punti con cui rappresentare la spline:

```
public Point[] calculatePoints(){
    //declaration of an array of points of dimension NUMPOINTS
    Point[] points = new Point[NUMPOINTS];
    double x;
    double y;
    double t;
    double xCubic;
    for (int i=0; i < NUMPOINTS; i++){
        //t is the index i normalized respect to NUMPOINTS
        t = (double) i / (NUMPOINTS - 1);
        //cubic distribution of points along intervalX
        xCubic = intervalX * Math.pow(t, 3);
        //addition of startPoint x component
        x = startPoint.getX() + xCubic;
        //y coordinate evaluation
        y = evaluateY(x);
        //add the calculated point to the array
        points[i] = new Point(x, y);
    }
    return points;
}
```

x è la variabile indipendente per la funzione spline, pertanto la distribuzione cubica viene effettuata su `intervalX`.

Implementazione metodo che restituisce l'array delle pendenze nei punti della spline:

```
public double[] calculateSlopes(){
    //declaration of an array of slopes of dimension NUMPOINTS
    double[] slopes = new double[NUMPOINTS];
    //linear segment
    if(splineFunction == null)
    {
        //angular coefficient of the linear segment
        double m = intervalY / intervalX;
        //constant slope on the linear segment
        for(int i=0; i < NUMPOINTS; i++){
            slopes[i] = Math.atan(m);
        }
        return slopes;
    }
    //interpolating spline function
    else{
        double x;
        double t;
        double xCubic;
```

```

        for (int i=0; i < NUMPOINTS; i++) {
            //t is the index i normalized respect to NUMPOINTS
            t = (double) i / (NUMPOINTS - 1);
            //cubic distribution of points along intervalX
            xCubic = intervalX * Math.pow(t, 3);
            //addition of startPoint x component
            x = startPoint.getX() + xCubic;
            //spline slope
            slopes[i] = Math.atan(splineFunction.derivative().value(x));
        }
        return slopes;
    }
}

```

Se `splineFunction` è null la pendenza è rappresentata dal coefficiente angolare della retta passante per i due punti di partenza e arrivo, ed è costante per tutti i punti del segmento lineare:

$$slopes[i] = \tan^{-1} m.$$

Altrimenti se `splineFunction` non è null, si calcola la derivata della spline con il metodo `derivative` di `PolynomialSplineFunction`. Dopodichè si ottiene il valore della derivata in un punto con il metodo `value`, che riceve come parametro la coordinata x del punto.

8.7.6. Simulazione

La simulazione della caduta delle masse lungo le curve è gestita dalla classe `SimulationManager`.

Definizione classe `SimulationManager` e dichiarazione dei campi:

```

public class SimulationManager {
    private Mass mass;
    private Curve curve;
    private Point[] points;
    private double[] slopes;
}

```

Ciascuna istanza di `SimulationManager` gestisce la simulazione della caduta di una massa lungo una curva. Per questo motivo, fra i suoi attributi figurano: la massa in caduta, la curva lungo la quale avviene la caduta, un array contenente i punti di tale curva e un array che memorizza, in corrispondenza dei punti precedenti, le pendenze della curva espresse in radianti.

Metodo che determina, per ogni punto della curva, l’istante di tempo (calcolato a partire dall’avvio della simulazione, cioè dall’inizio della caduta) in cui la massa si trova in quel punto durante il suo moto descendente:

```
//creates an array that associates to each point, the time in which the mass
passes on that point.
public double[] calculateTimeParametrization(double g) {
    times = new double[points.length];
    times[0] = 0; //time initialization in start point
    //initialization to avoid approximated integral to diverge to infinity
    times[1] = Double.MIN_VALUE;
    double h1; //y difference between point i and start point
    double h2; //y difference between point i+1 and start point
    double v1; //speed in point i
    double v2; //speed in point i+1
    double v1y; //y component of v1
    double v2y; //y component of v2
    double integrand; //function to be integrated
    double dy; //y difference between i point and i+1 point
    for (int i = 2; i < points.length; i++) {
        h1 = points[i-1].getY() - curve.getStartPoint().getY();
        // ensure the integral doesn't diverge to infinity if y difference
        is so small that is approximated to 0
        if(h1==0){
            times[i] = times[i-1] + Double.MIN_VALUE;
            continue;
        }
        h2 = points[i].getY() - curve.getStartPoint().getY();
        //expressions for velocity due to conservation of energy
        v1 = Math.sqrt(2*g*h1);
        v2 = Math.sqrt(2*g*h2);
        //velocity scomposition on y component
        v1y = v1*Math.abs(Math.sin(slopes[i-1]));
        v2y = v2*Math.abs(Math.sin(slopes[i]));
        dy = Math.abs(points[i].getY() - points[i-1].getY());
        integrand = ((1/v1y + 1/v2y)/2); //mean value for integrand
        //finite increment added to previous calculated time
        times[i] = times[i-1] + integrand * dy;
    }
    return times;
}
```

I tempi sono memorizzati in un array della stessa lunghezza dell’array dei punti della curva. L’associazione tra punto e relativo tempo è rappresentata dalla corrispondenza degli indici di posizione nei rispettivi array, allo stesso modo delle pendenze:

points[i] → slopes[i] → times[i].

Il primo elemento dell'array dei tempi viene inizializzato a 0 e rappresenta l'istante in cui la massa si trova nel punto di partenza e inizia a cadere.

I tempi per i punti successivi vengono calcolati con la tecnica di integrazione numerica con il metodo dei trapezi.

Metodo dei trapezi:

$$\int_a^b f(x)dx \cong \sum_{k=0}^N \frac{f(x_{k-1}) + f(x_k)}{2} \cdot (x_k - x_{k-1}).$$

Per il calcolo del tempo T_i , per un generico punto nella i -esima posizione dell'array dei punti della curva, supponendo la componente verticale della velocità v_Y funzione di y ($y_k = \text{points}[k].\text{getY}()$):

$$T_i = \int_{y_0}^{y_i} \frac{1}{v_Y(y)} dy \cong \sum_{k=1}^i \frac{\frac{1}{v_Y(y_{k-1})} + \frac{1}{v_Y(y_k)}}{2} \cdot (y_k - y_{k-1}).$$

In realtà la curva può anche risalire lungo y , e quindi v_Y non è funzione di y , perché possono esserci più punti distinti della curva con la stessa y , ma pendenze della curva in tali punti, e quindi v_Y , diverse. L'integrale numerico rimane comunque corretto se si considera la componente verticale della velocità in funzione dell'indice di posizione k di ciascun punto, invece che funzione di y_k . L'espressione diventa:

$$T_i \cong \sum_{k=1}^i \frac{\frac{1}{v_{Y_{k-1}}} + \frac{1}{v_{Y_k}}}{2} \cdot |y_k - y_{k-1}|.$$

Si nota che gli intervalli $y_k - y_{k-1}$ non sono necessariamente tutti della stessa ampiezza come nel classico metodo dei trapezi, ma hanno ampiezze diverse secondo la distribuzione dei punti dell'array `points`.

Gli intervalli $y_k - y_{k-1}$ devono essere interpretati come distanze lungo cui si calcola il tempo di percorrenza, e quindi devono necessariamente essere positivi. Tuttavia, ci sono possibilmente delle regioni in cui la curva risale lungo y :

$y_k < y_{k-1} \rightarrow y_k - y_{k-1} < 0$. Per questo motivo bisogna considerare il valore assoluto $|y_k - y_{k-1}|$.

Poiché è necessario calcolare il tempo per ogni punto della curva bisogna calcolare ciascun valore T_i per ogni i da 0 a `points.length-1`.

Il primo elemento dell'array viene inizializzato a 0 e rappresenta l'istante in cui la massa si trova nel punto di partenza e inizia a cadere. Viene inizializzato anche il secondo elemento dell'array al minimo valore reale rappresentabile. Questo perché il suo calcolo richiederebbe di considerare la situazione nel punto di partenza a velocità 0, il cui reciproco divergerebbe a infinito, generando un errore a tempo di esecuzione.

Per i punti successivi, il calcolo dei tempi si basa su un ciclo su i , da 0 a `points.length - 1`. Ad ogni iterazione viene sommato l'incremento al tempo precedente. In questo modo l'array dei tempi viene popolato progressivamente, calcolando ciascun nuovo valore a partire da quello precedente. Infatti, ogni valore di T_i si può calcolare da quello precedente T_{i-1} :

$$\begin{aligned} T_i &\cong \sum_{k=1}^i \frac{\frac{1}{v_{Yk-1}} + \frac{1}{v_{Yk}}}{2} \cdot |y_k - y_{k-1}| = \\ &= \sum_{k=1}^{i-1} \frac{\frac{1}{v_{Yk-1}} + \frac{1}{v_{Yk}}}{2} \cdot |y_k - y_{k-1}| + \frac{\frac{1}{v_{Yi-1}} + \frac{1}{v_{Yi}}}{2} \cdot |y_i - y_{i-1}| = \\ &= T_{i-1} + \frac{\frac{1}{v_{Yi-1}} + \frac{1}{v_{Yi}}}{2} \cdot |y_i - y_{i-1}|. \end{aligned}$$

Il modulo della velocità è dato da:

$$v_i = \sqrt{2g(y_i - y_0)}.$$

La componente verticale della velocità in un punto i si calcola con la pendenza in radianti della curva nel medesimo punto, precedentemente calcolata e memorizzata nell'array `slopes` (chiamando `slopes[i] = theta_i`).

$$v_{Yi} = v_i \cdot |\sin \theta_i|.$$

Della componente verticale della velocità si è interessati al valore assoluto.

Se $|y_k - y_{k-1}|$ è troppo piccolo, potrebbe essere approssimato a 0 nella rappresentazione digitale del valore reale. Tale situazione si può avere nei primi punti, in quanto sono molto addensati vicino al punto di partenza a causa della distribuzione cubica. Tale approssimazione causa problemi perché annulla la velocità, facendo divergere a infinito il tempo, causando quindi un errore a tempo di esecuzione. Per questo motivo si controlla

con un’istruzione condizionale se si verifica tale situazione e, in caso positivo, il valore per il tempo viene calcolato sommando il minimo valore reale rappresentabile al tempo precedente.

8.8. Testing

Oltre a svolgere i test d’unità con Junit per dimostrare la correttezza dei metodi, l’attività di testing si è basata principalmente sul test manuale: eseguendo la simulazione e interagendo con l’interfaccia.

Gli aspetti matematici sono stati valutati con il supporto di geogebra.

Un’importante prova della correttezza dei calcoli dei tempi di caduta è il fatto che la cicloide, cioè la soluzione del problema della brachistocrona, è effettivamente sempre la curva per cui il tempo di caduta è il minimo.

Il rilevamento dei bug nell’interfaccia è stato svolto facendo interagire con il sistema utenti non implicitamente a conoscenza del flusso dell’interazione, in modo da sottoporre comportamenti non previsti.

Tale approccio ha svelato, ad esempio, una scorretta gestione degli input dei punti intermedi da interpolare, che potevano essere inseriti anche dopo che la spline era stata calcolata e disegnata nel pannello.

8.9. Manutenzione

In seguito al completamento del prototipo è stata eseguita una corposa attività di ristrutturazione guidata dai tool:

- SonarLint ha guidato il miglioramento della qualità del codice
- StanIDE ha suggerito miglioramenti nella struttura modulare del sistema.

L’attività di manutenzione ha richiesto complessivamente uno sforzo circa pari a quello dello sviluppo.

La manutenzione perfettiva ha riguardato l’aggiunta della possibilità di scegliere tra diversi campi gravitazionali per selezionare l’accelerazione di gravità.

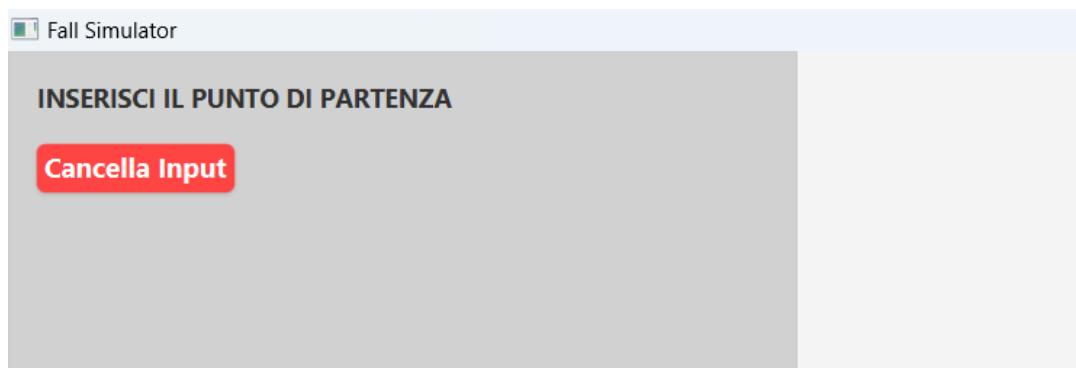
La manutenzione preventiva ha riguardato l'aggiunta di commenti per esplicare il significato del codice.

8.10. Demo

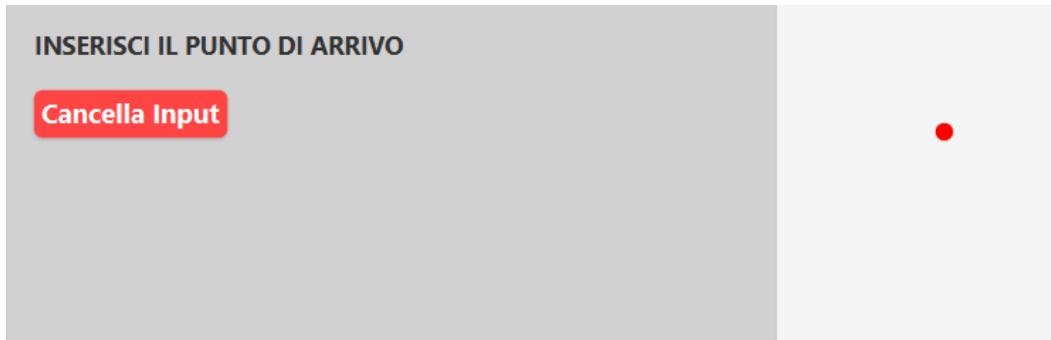
- Selezione dell'accelerazione di gravità



- Inserimento del punto di partenza



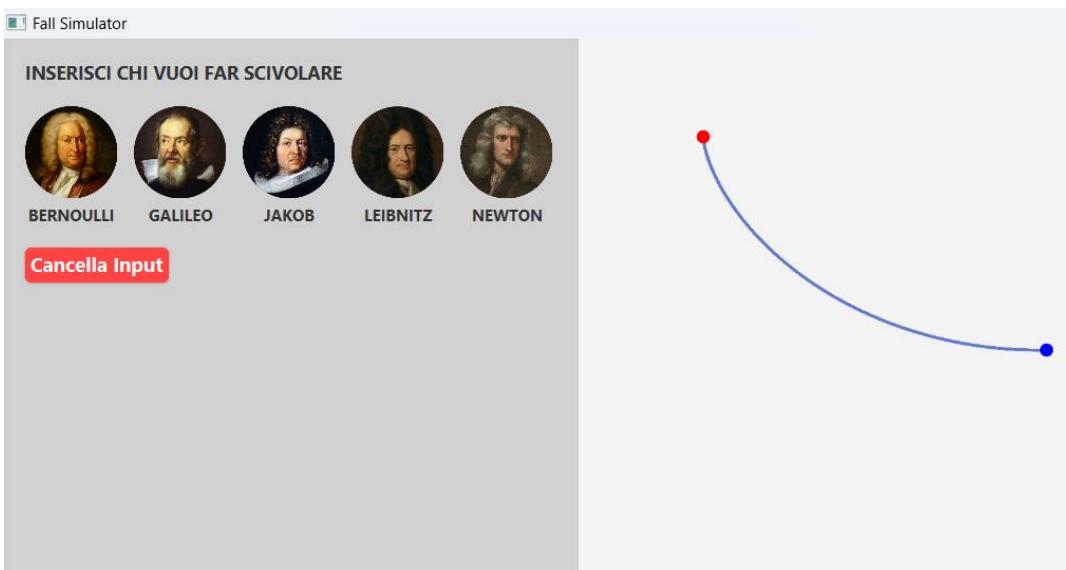
- Inserimento del punto di arrivo



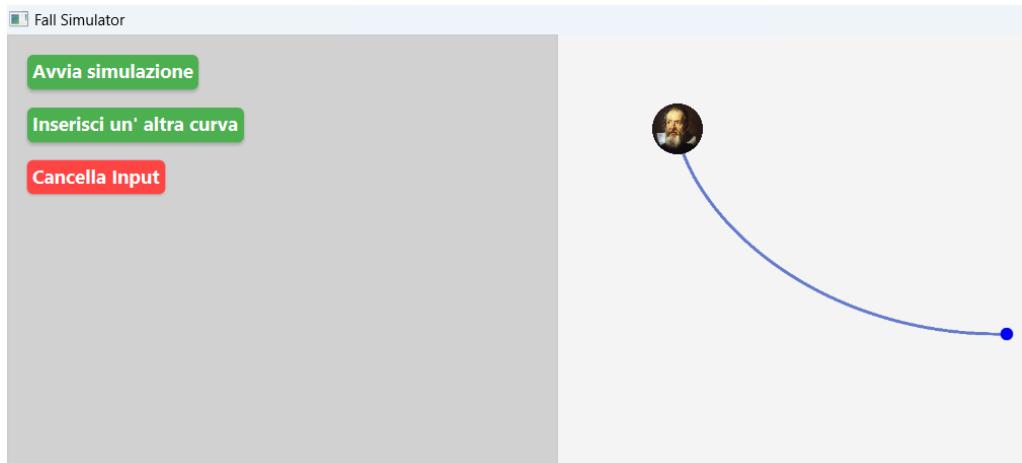
- Selezione della curva



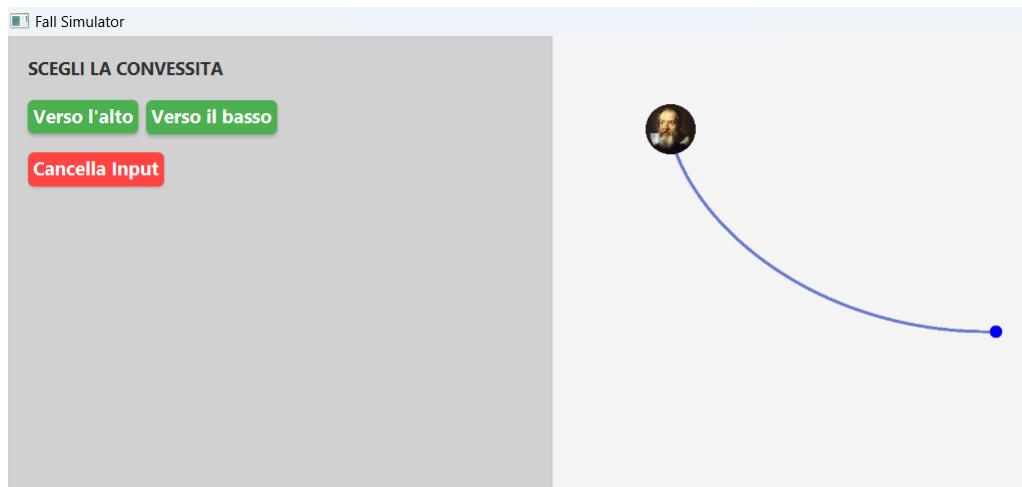
- Disegno della curva e selezione della massa



- Possibilità di inserire un'altra curva o avviare la simulazione



- Selezione della convessità della circonferenza



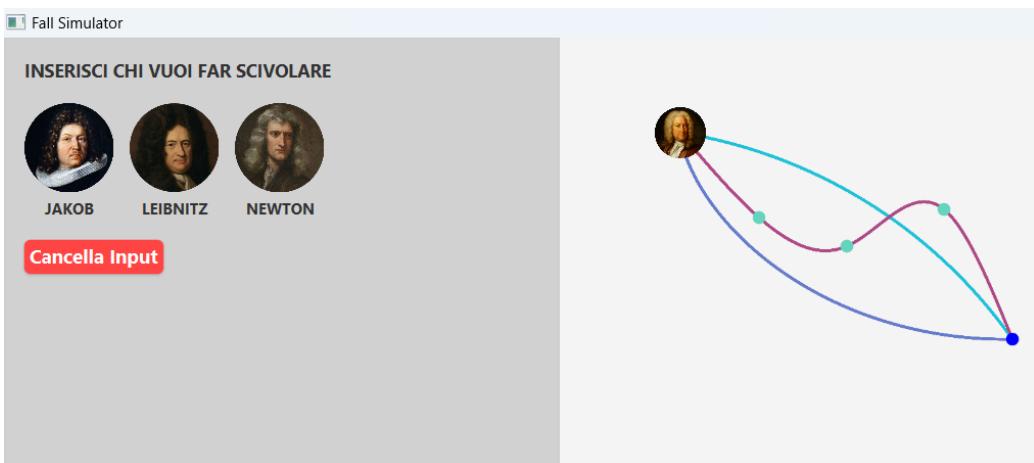
- Selezione del raggio della circonferenza



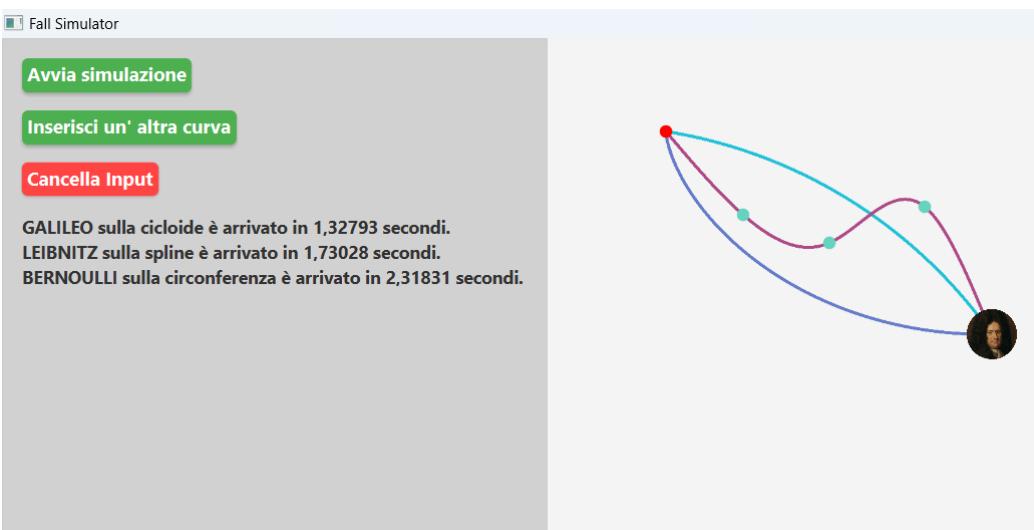
- Inserimento dei punti intermedi da interpolare



- Disegno della spline



- Simulazione e classifica dei tempi di arrivo



8.11. Sviluppi futuri

Un'estensione che potrebbe rappresentare una sfida futura, come occasione per imparare nuovi tecnologie e strumenti, è la realizzazione della simulazione come applicazione VR (Virtual Reality) con Unity.



Ciò richiederebbe una corposa fase di manutenzione adattiva per migrare il sistema su una diversa piattaforma, riscrivendo il codice in C#.



Inoltre, gli aspetti matematici andrebbero generalizzati nello spazio tridimensionale, invece che nel piano.

9. Conclusioni

Lo studio condotto nelle sezioni precedenti ha permesso di esplorare il problema della Brachistocrona da molteplici prospettive, combinando diversi approcci teorici con lo sviluppo di una simulazione software progettata per modellare la caduta dei gravi lungo curve. Quest'ultima conferma sperimentalmente che la soluzione del problema è un arco di cicloide, fornendo una verifica numerica ai risultati analitici. La capacità di visualizzare dinamicamente il moto lungo diverse curve rende concetti astratti tangibili, facilitando l'apprendimento. La realizzazione della simulazione ha rappresentato dunque un banco

di prova per validare i risultati ottenuti, traducendo la teoria in un prodotto ingegneristico concreto.

Il problema della brachistocrona rappresenta un ponte tra astrazione matematica e applicazione ingegneristica. Questo percorso, che parte da una sfida storica per giungere a una simulazione software, riflette non solo l’evoluzione delle tecniche matematiche, ma anche il processo di apprendimento tipico dell’ingegneria.

Il passaggio dalla formulazione teorica alla realizzazione del software ha richiesto l’integrazione di conoscenze provenienti da molteplici discipline: matematica, fisica, ingegneria e informatica. Ciò si è reso necessario per affrontare con successo tutti gli aspetti del problema: dalla modellazione delle curve e della dinamica della caduta, alla loro realizzazione nel codice, evidenziando la caratteristica fondamentale dell’approccio ingegneristico, in cui le discipline teoriche guidano la progettazione e l’implementazione tecnica.

La soluzione teorica del problema, rappresentata dalla cicloide, trova conferma nella simulazione sviluppata. L’implementazione software non solo verifica numericamente che la cicloide minimizza il tempo di caduta, ma dimostra anche come la simulazione, supportata da strumenti computazionali, permetta di analizzare modelli matematici complessi.

È stata delineata una panoramica completa e interdisciplinare del problema della brachistocrona, attraverso l’integrazione di metodi analitici e strumenti computazionali. L’obiettivo finale è quello di comprendere e ottimizzare i fenomeni fisici, un principio su cui si basa il progresso della scienza.

Il percorso che ha portato a questo risultato ha richiesto una lunga fase di graduale acquisizione di conoscenza dell’argomento: dal primo approccio al problema, al tentativo di risolvere il problema in modo autonomo, sfruttando le sole conoscenze di base acquisite fino a quel momento: trigonometria e calcolo differenziale e integrale. Con il tempo, l’approfondimento di teorie più avanzate ha permesso di comprendere le altre soluzioni moderne al problema. Tale approfondimento ha reso possibile l’implementazione degli aspetti matematici in un software simulativo. Lo sviluppo del

software FallSimulator ha rappresentato dunque il culmine di questo percorso. Ha richiesto di integrare alla matematica, conoscenze ingegneristiche come progettazione orientata agli oggetti, programmazione, realizzazione di interfacce grafiche e algoritmi numerici.

Trasformare un problema storico in un progetto ingegneristico, unendo intuizioni concettuali del passato con strumenti moderni, è stato un lavoro particolarmente gratificante, per la soddisfazione di aver dato nuova vita a una sfida che ha coinvolto illustri figure della storia della scienza.

Riferimenti bibliografici

- [1] M. de Icaza Herrera, "Galileo, Bernoulli, Leibniz and Newton around the brachistochrone problem" *Revista Mexicana de Física*, vol. 40, no. 3, pp. 59-75, 1994. [Online]. Disponibile: https://www.researchgate.net/publication/267660667_Galileo_Bernoulli_Leibniz_and_Newton_around_the_brachistochrone_problem
- [2] J. Bernoulli, "Problema novum ad cuius solutionem Mathematici invitantur" *Acta Eruditorum*, vol. 18, p. 269, 1696. [Online]. Disponibile: https://books.google.it/books?id=4q1RAAAcAAJ&pg=PA269&redir_esc=y#v=onepage&q&f=false.
- [3] Y. Nishiyama, "The Brachistochrone Curve: The Problem of Quickest Descent" *International Journal of Pure and Applied Mathematics*, vol. 82, no. 3, pp. 409–419, 2013. [Online]. Disponibile: <https://ijpam.eu/contents/2013-82-3/8/8.pdf>.
- [4] G. Buttazzo e M. Mintchev, "Curve brachistocrone in campi di gravità," *Quaderno del Dipartimento di Matematica dell'Università di Pisa*, vol. 4, pp. 1–20, 2003. [Online]. Disponibile: <https://cvgmt.sns.it/media/doc/paper/4/brachisto.pdf>.
- [5] G. Buttazzo, G. Dal Maso e E. De Giorgi, "Calcolo delle variazioni" Enciclopedia del Novecento, vol. XI Suppl. II, Istituto della Enciclopedia Italiana, Roma, pp. 831–848, 1998.
- [6] H. J. Sussmann and J. C. Willems, "The Brachistochrone Problem and Modern Control Theory" in *Contemporary Trends in Nonlinear Geometric Control Theory and Its Applications*, A. Anzaldo-Meneses, B. Bonnard, J. P. Gauthier, and F. Monroy-Pérez, Eds. Singapore: World Scientific Publishing, 2002, pp. 113–151. [Online]. Disponibile: https://doi.org/10.1142/9789812778079_0004.
- [7] D. E. Kirk, "Optimal Control Theory: An Introduction", Mineola, NY: Dover Publications, 2004.