Relazione di tesi: La serie di Fourier e una sua applicazione alla grafica

Matteo Bramardi, candidato Prof. Paolo Boggiatto, relatore

27 giugno 2020

Contents

Introduzione

L'idea per l'argomento della tesi nasce dal video *But what is a Fourier series? From heat flow to circle drawings* pubblicato su YouTube da Grant Sanderson, noto come ***3Blue1Brown***. Rimasto colpito dal fascino estetico delle animazioni e dalla loro squisita immediatezza, ho deciso di approfondirne i fondamenti teorici.

Il cuore della tesi è la **presentazione** che, oltre ad una dissertazione teorica del tema, ripropone le suddette **animazioni**, da me ricreate ed adattate. La presentazione è affinata per agevolare la comprensione attraverso l'**interazione** con diversi suoi elementi e aspira ad essere un esempio di applicazione delle tecnologie informatiche a fini didattici.

Essa è il risultato di quasi un anno di lavoro e studio personale e concretizza la mia idea di presentare la matematica in modo **coinvolgente** e con l'ausilio di **visualizzazioni grafiche** di forte impatto e semplice interpretazione.

Chapter 1

Fasi del lavoro

1.1 Primi prototipi

Dopola visione del video sopracitato nel mese di luglio 2019, ho realizzato il primo prototipo dell'animazione principale in *Javascript* seguendo alcuni tutorial di *The Coding Train*. A dicembre dello stesso anno ho poi riscritto il precedente prototipo in *Java* e ne ho ampliato le funzionalità. Ciò mi ha permesso di familiarizzare con i vari algoritmi impiegati nella generazione delle informazioni utili alla visualizzazione dell'animazione e la loro implementazione.

1.2 Ricerca teorica

Già nel mese di dicembre 2019 mi ero approcciato allo studio della trasformata di Fourier e di una sua applicazione alla compressione delle immagini con il metodo JPEG *lossy* - basato sulla trasformata discreta del coseno - con lo scopo di produrre un trittico di poster sull'argomento, parte della prova di esame del corso di *Analisi Matematica 3*.

La fase di ricerca teorica vera e propria ha avuto inizio nel mese di marzo 2020 e, su indicazione del mio relatore Prof. Paolo Boggiatto, ha riguardato principalmente lo studio approfondito di alcune sezioni del libro ***Fourier Analysis and Applications*** di C. Gasquet e P. Witomski: nello specifico, il capitolo II - *i segnali periodici e la serie di Fourier* -, la lezione 8 del capitolo III - *la trasformata discreta di Fourier* - e le lezioni 17, 18 e 19 del capitolo VI - *la trasformata di Fourier di funzioni integrabili, la trasformata inversa e lo spazio* $\mathcal{S}(\mathbb{R})$.

La questione centrale della tesi, alla base del funzionamento dell'animazione, riguarda la possibilità di scrivere una funzione $f : \mathbb{R} \longrightarrow \mathbb{C}$ periodica di periodo

a come

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n e^{2\pi i n \frac{t}{a}} \ ,$$

dove $c_n \in \mathbb{C}$ e $n \in \mathbb{Z}$, sotto *minime ipotesi* sulla funzione f.

Inoltre, durante lo sviluppo del software, si è rivelato utile affrontare il tema delle curve di Bèzier, in particolare delle *polybézier* - ovvero di una curva di Bèzier continua definita a tratti. Per potenziare le mie conoscenze in materia, ho fatto uso del libro *A Primer on Bézier Curves*, disponibile online, che mi ha fornito le basi necessarie per manipolare tali oggetti da un punto di vista informatico.

Prima della fase di programmazione si è poi resa necessaria una ricerca degli strumenti informatici più adatti. Una discussione degli strumenti adottati e del loro impiego è consultabile al capitolo dedicato…

1.3 Animazioni

Terminata la raccolta e lo studio delle informazioni necessarie, ha potuto avere inizio la programmazione delle animazioni in maniera più accurata rispetto ai primi prototipi. Il loro sviluppo è avvenuto in *JavaScript*, facendo ampio uso delle librerie *open source p5.js*, per il *creative coding*, e *complex.js*, per la gestione dei numeri complessi. Ciò ha permesso, a differenza di un video, di rendere le animazioni **interattive**, andando così a potenziare il legame con lo spettatore che si trasforma quindi in utente attivo.

L'obiettivo delle animazioni è quello di agevolare la comprensione dei concetti teorici tramite una loro **visualizzazione immediata** e metterne in evidenza le sfaccettature più nascoste, andando ad affiancare ed assistere l'ingegno e le capacità immaginative dei loro fruitori.

Molta attenzione è stata riposta nella loro **estetica**, dacché ritengo che l'aspetto artistico non sia affatto secondario al contenuto, ma piuttosto funzionale a quest'ultimo, poiché in grado di avvicinare e coinvolgere lo spettatore-utente instaurando una connessione emotiva e sensoriale più forte. La filosofia alla base di questo approccio deriva sicuramente dalla visione artistica del video originale. Tuttavia, è stata ampiamente influenzata anche dagli *Elementi* di Euclide a colori, ovvero un'edizione del celebre libro di geometria in cui simboli e figure colorate sostituiscono le lettere, e dal pensiero del suo autore, Oliver Byrne, che ho avuto modo di apprezzare durante il corso di *Storia della Matematica Antica e Moderna*.

Ho infine curato l'**accessibilità**, ovvero la facilità di utilizzo, e la **portabilità**, cioè la corretta visualizzazione e fruizione su diversi tipi di dispositivi, principi chiavi per garantire all'utente un'esperienza fluida e priva di ostacoli o barriere di carattere tecnico.

1.4 Presentazione

La presentazione è stata realizzata con ***Reveal.js***, un framework *open source* per presentazioni HTML a cui ho apportato diverse personalizzazioni. Ciò mi ha permesso di **integrare perfettamente** (ovvero *seamlessly* in inglese) le **animazioni** nell'ambiente della presentazione, senza richiedere all'utente operazioni aggiuntive. Inoltre, il file HTML può essere visualizzato da un'ampia gamma di dispositivi tramite un qualsiasi *browser* e le ridotte dimensioni del file lo rendono ideale per la distribuzione online.

La filosofia che mi ha guidato nella creazione delle slide è la stessa discussa nel precedente paragrafo: accessibilità e piacevolezza estetica hanno costituito il fondamento su cui sviluppare il contenuto e il mezzo con cui esaltarlo, ma senza in alcun modo prevaricarlo. In tal senso, diversi accorgimenti sono stati adottati per affinare l'esperienza dell'utente finale.

Nel capitolo successivo verrà fornito un commento di carattere teorico alla presentazione.

Chapter 2

Commento alla presentazione

La presentazione segue per sommi capi le prime due lezioni del **capitolo II** del libro ***Fourier Analysis and Applications*** di C. Gasquet e P. Witomski, oggetto della ricerca. Le slide sono autonome da un punto di vista teorico e possono essere utilizzate parallelamente al libro sopracitato come strumento supplementare per l'apprendimento.

Funzionalità

- Le animazioni sono **integrate** nella presentazione ed è possibile **interagire** con esse;
- formule, teoremi (o corollari) ed esempi sono demarcati rispettivamente con colore azzurro, verde e giallo per un più **semplice riconoscimento**;
- formule e teoremi vengono visualizzati in una piccola scheda al passaggio o click del mouse sui rispettivi riferimenti, rendendo **immediata** la loro **consultazione**, mentre un ulteriore click su tali schede conduce alla slide in cui sono stati introdotti;

2.1 Segnali trigonometrici

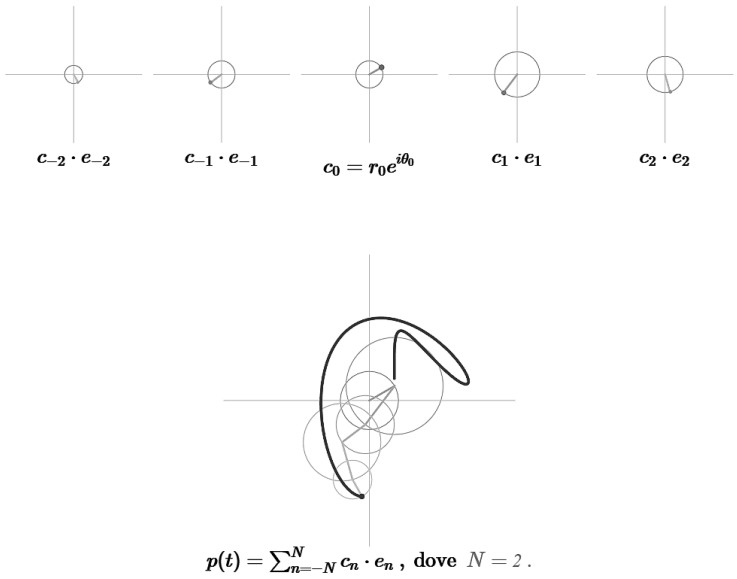
Diciamo ***polinomi trigonometrici*** le funzioni del tipo

$$p(t) = \sum_{n=-N}^{N} c_n e^{2i\pi n \frac{t}{a}} \ , \tag{2.1}$$

dove $a, t \in \mathbb{R},\ c_n \in \mathbb{C}$. $p(t)$ ha periodo a e grado minore o uguale a N.

Seguono immediatamente **due animazioni** per visualizzare un polinomio trigonometrico.

- Nella prima è possibile - tramite *rotella del mouse* o *trascinamento verticale* - **costruire progressivamente** un polinomio trigonometrico aumentandone il grado e, di conseguenza, il numero di addendi. Il primo elemento visibile è il coefficiente c_0, ovvero un numero complesso. Gli altri elementi sono costituiti dal coefficiente complesso c_n moltiplicato per $e_n(t) = e^{2\pi i n \frac{t}{a}}$ ovvero la curva chiusa complessa con supporto la *circonferenza unitaria*, percorsa con frequenza n. Poiché la moltiplicazione di due numeri complessi può essere interpretata come il multiplo di una rotazione, $c_n \cdot e_n(t)$ rappresenta la curva chiusa con supporto una circonferenza di *raggio* $|c_n|$ e una *fase* $arg(c_n) = \theta_n$ (dove $c_n = r_n e^{i\theta_n}$) rispetto alla curva $|c_n| \cdot e_n(t)$. Le varie componenti del tipo $c_n \cdot e_n(t)$ possono essere sommate con la definizione usuale di somma in campo complesso, ottenendo una curva che rappresenta proprio il polinomio trigonometrico.



- Nella seconda animazione è possibile - sempre tramite *rotella del mouse* o *trascinamento verticale* - rivelare l'asse della variabile t per mezzo di una rotazione tridimensionale, visualizzando il polinomio trigonometrico nello spazio $\mathbb{R} \times \mathbb{C}$.

Introduciamo quindi T_n, spazio vettoriale dei polinomi trigonometrici $p(t)$ di grado minore o uguale a N, dotato del prodotto scalare

$$(p,q) = \int_0^a p(t)\overline{q}(t)dt \ .$$

Da $(p,n) = ac_n$ ricaviamo quindi la ***formula di Fourier***

$$c_n = \frac{1}{a}\int_0^a p(t)e^{-2i\pi n\frac{t}{a}}dt \ . \tag{2.2}$$

Si pone quindi la **domanda fondamentale**:

> se $f : \mathbb{R} \longrightarrow \mathbb{C}$ è una funzione arbitraria di periodo a, possiamo trovare una decomposizione di f della forma
>
> $$f(t) = \sum c_n e^{2i\pi n\frac{t}{a}} \ , \tag{2.3}$$
>
> sotto minime ipotesi su f ?

2.2 Segnali periodici e serie di Fourier

In un celebre articolo del 1807, **Joseph Fourier** afferma che la risposta a tale domanda sia affermativa, a patto siano consentite somme infinite.

È possibile ridefinire questa risposta con gli strumenti della matematica moderna. Per farlo introduciamo lo spazio

$$L^2_p(0,a) = \left\{f : \mathbb{R} \longrightarrow \mathbb{C} \ : \ f \ \text{ ha periodo } a \text{ e } \int_0^a |f(t)|^2dt < -\infty\right\}$$

che, dotato delle usuali operazioni, è uno **spazio vettoriale**. Definiamo quindi il *prodotto scalare*

$$(f,g) = \int_0^a f(t)\overline{g}(t)dt \ ,$$

e la *norma* associata

$$\|f\|_2 = \left(\int_0^a |f(t)|^2 dt \right)^{\frac{1}{2}}$$

Per rispondere alla *domanda fondamentale* occorre trovare l'elemento f_N nel sottospazio T_n di $L_p^2(0,a)$ che ha la minima distanza da f. Se esiste, lo chiamiamo la ***miglior approssimazione*** di f in T_n. La soluzione è fornita dal seguente teorema:

Teorema

Esiste un unico polinomio trigonometrico f_N in T_n tale che

$$\|f - f_N\|_2 = \min_{p \in T_N} \|f - p\|_2$$

Questo polinomio è dato da

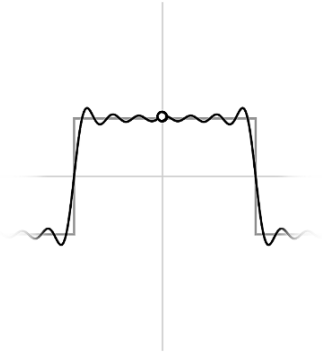
$$f_N(t) = \sum_{n=-N}^{N} c_n e^{2\pi i n \frac{t}{a}} \ , \tag{2.4}$$

dove

$$c_n = \frac{1}{a} \int_0^a p(t) e^{-2i\pi n \frac{t}{a}} dt \ . \tag{2.5}$$

Il passaggio successivo consiste nello studio della **convergenza**. Grazie ad un **esempio animato**, è possibile osservare come l'approssimazione, data da una somma di seni, tenda a alla funzione *onda quadra*:

- è possibile - tramite *rotella del mouse* o *trascinamento verticale* - aumentare in numero di addendi nella somma e, quindi, migliorare l'approssimazione.



In conclusione, è possibile scrivere:

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n e^{2i\pi n \frac{t}{a}} \ .$$

Si noti che questa è un'equivalenza nella norma di $L^2_p(0,a)$ e **non** significa che, per qualunque valore di t, il $f(t)$ sia uguale alla somma della serie.

2.3 Rappresentazione puntuale

Chapter 3

Software

Questo capitolo sarà dedicato ad una discussione dettagliata di alcune delle più interessanti tecniche informatiche e dei programmi impiegati nella la realizzazione della presentazione e delle animazioni.

Il **codice** è interamente disponibile su ***GitHub***.

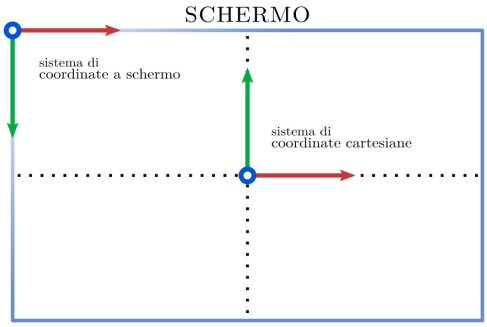
3.1 Animazioni

Le animazioni sono scritte in *JavaScript* e integrate in una pagina HTML. Per facilitare e velocizzare lo sviluppo, ho fatto ampio uso di ***p5.js***, una libreria grafica *open source* per il *creative coding*, che offre una varietà di funzioni per la rappresentazione di **costruzioni geometriche**, il *rendering* e la manipolazione del DOM.

Ho inoltre implementato diverse **funzionalità** per ottimizzare la gestione di alcuni aspetti delle animazioni.

3.1.1 Sistema di coordinate

Data la peculiarità del **sistema di coordinate a schermo** utilizzato da *p5.js* (contraddistinto dall'origine posizionata nell'angolo in alto a sinistra dello schermo e dall'asse y invertito, come schematizzato in figura) si è reso necessario trovare un metodo per gestire la conversione in **sistema di coordinate cartesiane**. Quest'ultimo viene **centrato** e **scalato** in base all'altezza e alla larghezza della finestra contenitore e riconfigurato al ridimensionamento della stessa.



Le funzione *toCartesian* - responsabile della conversione da coordiate a schermo a coordinate cartesiane - tiene conto della dimensione della finestra, informazione contenuta in *descaleFactor*, e della posizione del centro della finestra:

```
function toCartesian(x, y) {
  let cx = (x - xOrigin) * descaleFactor;
  let cy = (yOrigin - y) * descaleFactor;
  return createVector(cx, cy);
}
```

Lo stesso vale per la funzione inversa, che converte le coordinate cartesiane in coordinate a schermo:

```
function toScreenCoord(x, y) {
  let sx = x * scaleFactor + xOrigin;
  let sy = yOrigin - y * scaleFactor;
  return createVector(sx, sy);
}
```

3.1.2 Barra del progresso

La **barra del progresso** (o *progress bar*) è un'importante strumento di **accessibilità** e restituisce un *feedback* in merito al progresso possibile nell'interazione da parte dell'utente con l'animazione. La barra è posizionata sul lato destro dello schermo e monitora il comportamento dell'utente, attivandosi solo durante un interazione e scomparendo dopo qualche secondo, per non interferire con il resto dell'animazione.

È inoltre accompagnata da un'**icona** di un **mouse**, da me disegnata, che comunica all'utente, con delle apposite frecce pulsati, le azioni possibili.

3.1.3 La serie di Fourier

Per rappresentare la serie di Fourier di una funzione periodica $f(t)$ occorre **definire** con precisione la **funzione** - in modo che sia anche semplice da replicare graficamente - e calcolarne i **coefficienti di Fourier**.

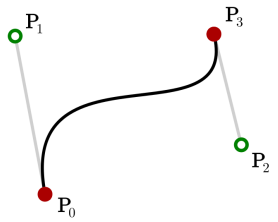
3.1.3.1 Polybézier

Il mezzo ideale per definire la funzione $f(t)$ si è rivelato essere la *polybézier*.

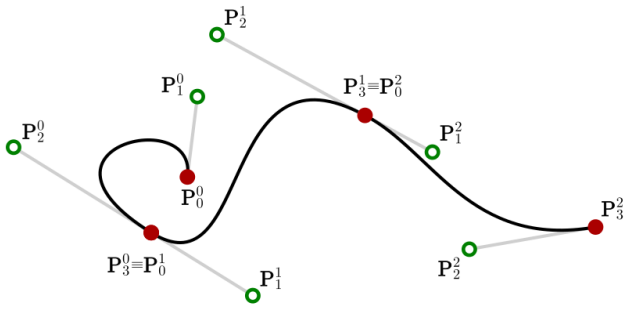
Una ***polybézier***, o curva di Bézier composta, è una **curva di Bézier** definita a tratti e continua, vale a dire una concatenazione di curve di Bézier. In particolare, si prenderà in esame il caso di una concatenazione di curve di Bézier **cubiche**, ovvero il tracciato percorso dalla curva di parametrizzazione

$$\mathbf{B}(t) = \mathbf{P}_0(1-t)^3 + 3\mathbf{P}_1 t(1-t)^2 + 2\mathbf{P}_2 t^2(1-t) + \mathbf{P}_3 t^3 \ , \ t \in [0,1] \ . \qquad (3.1)$$

La curva passa per i punti $\mathbf{P}_0$ e $\mathbf{P}_3$, ma non per i punti $\mathbf{P}_1$ e $\mathbf{P}$ che sono di *controllo* e forniscono informazioni direzionali. Il modo più semplice per comprendere la sua natura è visualizzarla:



Una *polybézier* cubica è formata da un numero $M \in \mathbb{N}$ di curve $\mathbf{B}^j(t),\ t \in [0,1],\ j = 0, ..., M-1$ tali che $\mathbf{B}^j(1) = \mathbf{B}^{j+1}(0)$. La curva risulterà chiusa se $\mathbf{B}^0(0) = \mathbf{B}^{M-1}(1)$.



Ho disegnate le curve con ***Inkscape***, un editor grafico vettoriale *open source*. Il file generato - di formato SVG - può essere aperto con semplice editor di testo e, al suo interno, ho identificato le coordinate dei punti. Ho poi scritto delle funzioni *JavaScript* per trasformare una stringa testuale in coordinate numeriche, **trovare le coordinate** di un punto $\mathbf{P}(t)$ giacente sulla curva dato t

```
getPoint(p0, p1, p2, p3, t) {
  let x = p0.x * pow(1 - t, 3) + 3 * p1.x * t * pow(1 - t, 2)
    + 3 * p2.x * pow(t, 2) * (1 - t) + p3.x * pow(t, 3);
  let y = p0.y * pow(1 - t, 3) + 3 * p1.y * t * pow(1 - t, 2)
    + 3 * p2.y * pow(t, 2) * (1 - t) + p3.y * pow(t, 3);
  return new Complex(x, y);
}
```

e infine **campionare** N **punti** lungo la *polybézier*, **distanziati equamente** nella variabile $t \in [0, 1]$:

```
samplePoints(N) {
  let s = 1.000000 / N;
  let M = this.points.length / 3;
  let sampledPoints = [];

  for (let j = 0; j < M; j++) {
    let points = {
      p0: this.points[3 * j], p1: this.points[1 + 3 * j],
      p2: this.points[2 + 3 * j], p3: this.points[(3 + 3 * j)
        % this.points.length]
    };
    for (let k = 0; k < N; k++) {
      sampledPoints[k + j * N] = this.getPoint(points.p0, points.p1,
```

```
          points.p2, points.p3, k * s);
    }
  }
  return sampledPoints;
}
```

3.1.4 La trasformata discreta di Fourier

Non potendo computare esplicitamente i **coefficienti di Fourier** con l'integrale

$$c_n = \frac{1}{a}\int_0^a f(t)e^{-2i\pi n\frac{t}{a}}dt \ ,$$

sono ricorso alla **trasformata discreta di Fourier**. Per farlo, è necessario assumere che la funzione $f(t)$ di cui si vogliono calcolare i coefficienti sia periodica di periodo a e che N dei suoi valori siano equamente distanziati sul periodo:

$$f\left(k\frac{a}{N}\right) = y_k \ , \quad k = 0, 1, 2, ..., N-1 \ .$$

Pertanto, si assume che il segnale $f(t)$ sia campionato a tempi equamente distanziati di $\frac{a}{N}$ unità. Si assume inoltre che la serie di Fourier converga puntualmente a f e che nei punti di discontinuità

$$f(t) = \frac{1}{2}(f(t+) + f(t-)) \ .$$

Con la *formula del trapezio*, si ottiene un'**approssimazione** dei coefficienti di Fourier della funzione f:

$$c'_n = \frac{1}{N}\sum_{k=0}^{N-1} y_k e^{-2\pi i n\frac{k}{N}} \tag{3.2}$$

Tradotto in termini informatici, la funzione responsabile del computo della trasformata discreta può essere così espressa:

```
function dft(y) {
  const Y = [];
  const N = y.length;

  for (let n = 0; n < N; n++) {
    let sum = new Complex(0, 0);
    for (let k = 0; k < N; k++) {
      const phi = (- TWO_PI * k * n) / N;
```

```
      const c = new Complex(cos(phi), sin(phi));
      sum.add(y[k].mult(c));
    }
    sum = sum.div(N);

    let freq = n;
    let amp = sum.amp();
    let phase = sum.phase();

    Y[n] = { re: sum.re, im: sum.im, freq, amp, phase };
  }

  return Y;
}
```

Applicando questo risultato ad una *polybézier* chiusa e composta da M curve di Bézier cubiche $\mathbf{B}^j(t),\ t \in [0,1],\ j = 0, ..., M-1$, si ottiene:

$$c'_n = \frac{1}{M \cdot N'} \sum_{j=0}^{M-1} \sum_{k=0}^{N'-1} \mathbf{B}^j\left(\frac{k}{N'}\right) e^{-2\pi i n \frac{j \cdot N' + k}{M \cdot N}} \tag{3.3}$$

dove N' è il numero di punti campionati per ogni curva di Bézier che compone la *polybébezier*.

3.1.5 La rappresentazione della serie di Fourier

Data la funzione $f(t)$, ovvero la *polybézier*, e i coefficienti della serie di Fourier associata, computati con una trasformata discreta di Fourier, si può procedere alla **rappresentazione grafica** della serie. Una funzione si occupa di calcolare i punti del percorso tracciato al variare del tempo ($time \in [0, 2\pi]$) e dati N coefficienti:

```
function epicycles(time) {
  let N = fourier.length;

  let x = 0;
  let y = 0;

  for (let n = 0; n < N; n++) {
    let prevx = x;
    let prevy = y;

    let freq = fourier[n].freq;
```

```
    let radius = fourier[n].amp;
    let phase = fourier[n].phase;

    x += radius * cos(freq * time + phase);
    y += radius * sin(freq * time + phase);

    ellipse(prevx, prevy, radius * 2);
    line(prevx, prevy, x, y);
  }
  return createVector(x, y);
}
```

I **punti** sono quindi **salvati** in un vettore (*path*) e **uniti** per formare il percorso:

```
for (let i = 0; i < path.length - 1; i ++) {
  line(path[i].x, path[i].y, path[i+1].x, path[i+1].y);
}
```

3.2 Presentazione

Come già spiegato in un precedente capitolo, le slide sono state create attraverso il *framework* per presentazioni HTML ***Reveal.js***, modificato e **personalizzato** per meglio adattarsi alla visione artistica generale del progetto. *Reveal.js* permette di scrivere la presentazione usando il noto linguaggio di *markup* HTML. Questo approccio offre numerosi vantaggi: la *portabilità* del file su molti dispositivi, la compattezza del file e la possibilità di una semplice diffusione online. Inoltre, l'aspetto estetico può essere controllato tramite un apposito *style sheet* CSS in modo rapido e preciso.

La scelta è ricaduta su questo approccio per la necessità di **integrare fluidamente** le animazioni: *Reveal.js* consente infatti di visualizzare pagine web sullo sfondo delle slide: ciò mi ha permesso di collegare i file HTML contententi le animazioni.

```
<!-- Convergenza -->
<section
  data-background-iframe="https://bradwave.github.io/thesis/animations/s4-animation.html"
        data-preload data-auto-animate>
        <h4>2.2.1 Convergenza dell'approssimazione $ $</h4>
        <hr>
        <p style="margin-top: -1%;">
                Cosa succede a $f_N$ per $N \rightarrow + \infty$ ?
        </p>
        <blockquote class="example" style="width: 80%; margin-top: 5px;
```

```
          margin-bottom: 5px; position: relative;" data-id="block">
                <div class="fragment fade-down" data-fragment-index="2"
                        style="position: absolute;
                          margin-left: auto; margin-right: auto; left: 0; right: 0;">
                        $$ \text{approssimata da: } \ f_N(t)=\frac{4}{\pi} \left( sin(t
                        \frac{1}{3}sin(3t) +
                        \frac{1}{3}sin(5t) + ...
                        \right) $$
                </div>
                <div class="fragment fade-down" data-fragment-index="1">
                        <div class="fragment fade-out" data-fragment-index="2"
                                style="position: relative;
                                margin-left: auto; margin-right: auto; left: 0; right:
                                $$ \textit{Esempio: } \ f(t) =
                                \begin{cases}
                                +1, & \text{se $ \ 0 \le t \lt \pi $} \\
                                -1, & \text{se $ \ \pi \le t \lt 2 \pi $}
                                \end{cases} $$
                        </div>
                </div>
        </blockquote>
</section>
```

Come visibile in questo esempio...

Le formule matematiche vengono visualizzate grazie al *display engine* ***MathJax***.