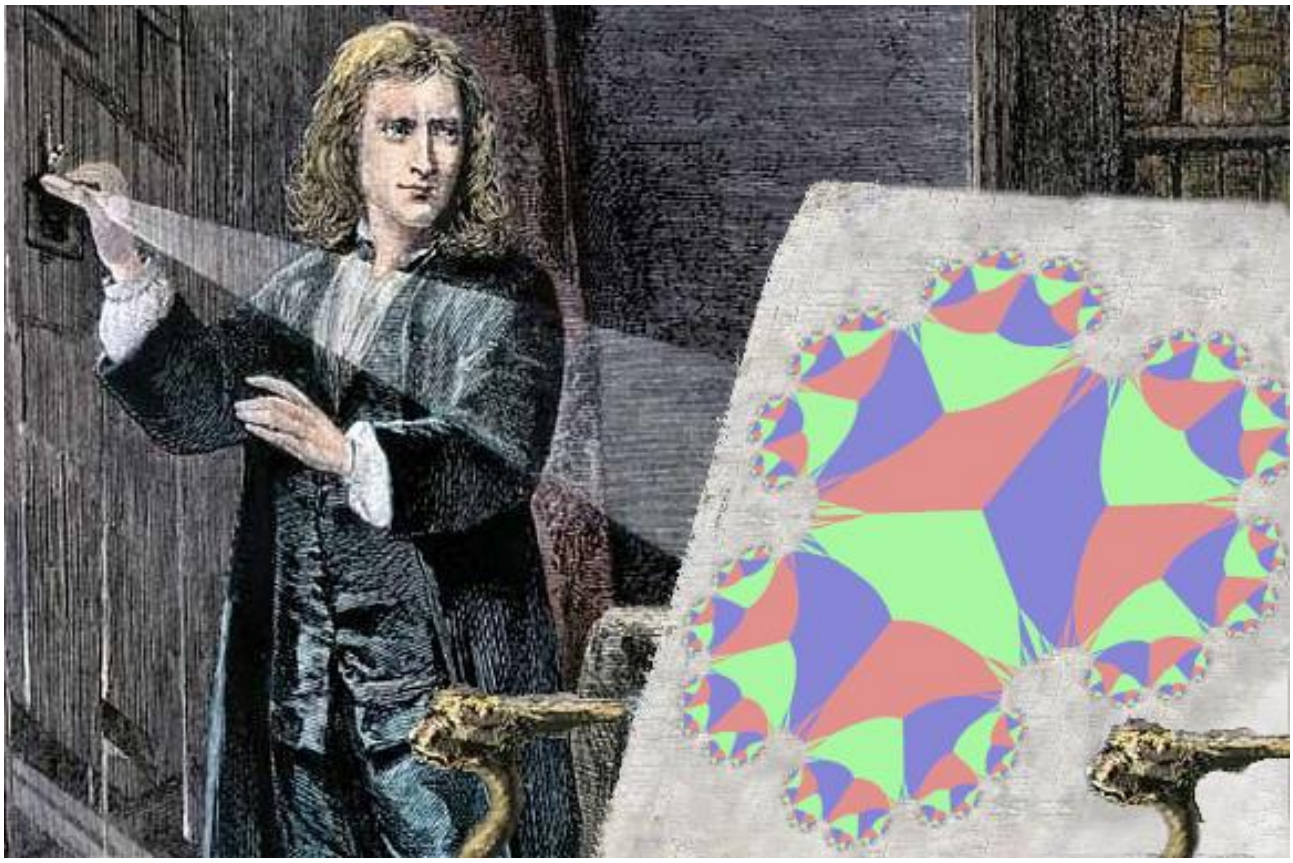
Newton en colors

El metode de Newton i les fractals



Markov

ÍNDEX

0.- *ABSTRACT*

This *Treball de Recerca* tries to explain, from a mathematical point of view, fractal structures. It is focused on those which come from iterating processes (iteration stands for repetition). And one specific iterating process is applying Newton's method to a polynomial. But in order to see some fractal results we ought to look at the complex plane, where Real and Imaginary numbers meet.

The aim of this *TdR* is to code two main programs: one that "draws" these fractal structures given a polynomial (whose roots are known) and another that finds the roots of any polynomial using Newton's method.

But this *TdR* will not start with coding. First of all, it gives a historical introduction, to know when were all the concepts that are going to appear first discovered, which leads to the most recent discoveries about the topic. Then the next point is about the necessary concepts to understand the topic, that not every reader may be aware of.

Given that solid base, it will explain the Newton's method and how it is applied with Real numbers, as well as the principles of iteration.

Next, it explains how arithmetic and iteration works with complex numbers and on the complex plane. Once this is done, we can apply Newton's method and see the results: by definition, there are fractal structures hidden between the basins of attraction.

The rest is about building up the programs and figuring out how to work with some extra theorems to find the roots.

Este trabajo intenta explicar desde un punto de vista matemático qué es una estructura fractal. Se centra en los fractales generados a partir de procesos iterativos (iterar significa repetir), en especial en el proceso de iteración resultante de aplicar el método de Newton a un polinomio. Pero para ello tendremos que hacerlo en el plano complejo, donde los números reales e imaginarios coexisten.

Para ello, habrá que explicar conceptos tales como números complejos, iteración, método de Newton y estructura fractal con el objetivo de hacer dos programas. Uno que "dibuje" estos fractales dado un polinomio de raíces conocidas y otro que permita encontrar las raíces de cualquier polinomio usando el método de Newton.

Pero no se empezará a programar desde cero. Primero se hará una introducción histórica para situar las ramas de la matemática de las cuales se hablarán. Lo siguiente es empezar a repasar (o ampliar) conceptos, con los cuales el lector puede estar más o menos familiarizado.

Con esa base de conocimientos, se podrá hablar del método de Newton y su iteración en en los números reales.

A continuación, se hablará de la aritmética de números complejos y de cómo funciona su iteración. Finalmente, de cómo aplicar el método en estos números y su resultado: por definición, hay estructuras fractales escondidas entre sus cuencas de atracción.

El resto tratará de ir construyendo los programas, con un teorema extra para encontrar las raíces.

1.- INTRODUCCIÓ

1.1 Presentació del tema

Heus aquí altre cop el nom de qui per molts és considerat el geni més gran de tots els temps, tema que no es rebatrà ni confirmarà; científic prolífic, polifacètic i imperant, en aquest aspecte hi ha consens; que tants maldecaps ha donat a estudiants, de secundària fins a darrer any de carrera, quan s'havien d'estudiar les lleis, identitats, equacions, el binomi o el mètode (entre moltes altres homonímies) que se li atribueixen.

Doncs aquí el tenim involucrat altre cop, a mode potser de segell de qualitat o de pedigrí, en un treball més. Aquest cop, però, utilitzarem la seva aportació amb un propòsit molt diferent del que ell havia pensat. El científic anglès el va concebre *ab initio* per a trobar aproximacions d'arrels de polinomis. De fet, per Newton era poc més que una casualitat algebraica, i no li va donar molta importància. En el que ell sí que confiava, era que les eines que incorporava el mètode serien revolucionàries: les derivades. I tan encertat va estar que segles més tard encara s'estudia el càlcul infinitesimal.

Res més lluny, aquí es fa servir per a obtenir fractals sobre el pla complex. Un gir argumental inesperat? Bé, em plau veure que hi hagi hagut estudiosos inquiets, no conformes amb els propòsits donats, i disposats a reutilitzar, de manera descarada, qualsevol coneixement anterior. És així com es fan els descobriments més interessants, davant la incertesa i a base de fer proves.

"Hi ha dos possibles finals: si el resultat confirma la hipòtesi, has fet una mesura. Si el resultat contradiu la hipòtesi, aleshores has fet un descobriment!" (Enrico Fermi). Tal com afirma aquesta cita, quan es poden preveure els resultats, solament es fan mesures per a corroborar una hipòtesi; i per a què els resultats siguin poc previsibles, cal que el procés sigui innovador. Així doncs, és necessari deixar de banda l'estudi convencional per a obrir nous camins.

I en matèria de poc convencional, hem de mirar obligadament el macroscòpic i el microscòpic. Les fractals segurament siguin la barreja dels dos mons: calen infinits processos (iteracions, en aquest cas) per a arribar a tenir detalls geomètrics infinitesimals, però de bellesa immensa.

1.2 Motivacions

Aquí he de sincerar-me. No va ser pas *ab origene* que vaig escollir aquest tema com a eix del treball. Estic ben segur que el lector haurà viscut algun moment la incertesa, el no saber què triar (i, després de fer-ho, no estar segur de si va pel bon camí).

La cosa va anar així (o almenys així ho vull explicar): en un principi, tenia la intenció de fer el treball sobre *pènduls*. La física és una matèria que admiro més que res per la vessant pràctica i aplicable en moltes ciències... però al final vaig renunciar-hi. *Secundo* tenia en ment la idea de centrar-me solament en *fractals*. Fer-ne un estudi dels més importants, embadalir-se amb les estructures infinites i intrigants que presenten... altre cop, *deficere.*

Ambdós temes presentaven darrere l'oportunitat d'escriure un programa que simulés o bé el comportament real d'un pèndul (en un entorn amb forces i gravetat) o bé una aproximació del fractal (després de processos geomètrics i moltes iteracions). Aquest era el meu principal mòbil: la programació. Però així com la física em causa admiració i la informàtica motivació, la meva passió oculta són les matemàtiques.

I aleshores vaig trobar-me, com qui ensopega amb la seva cançó favorita canviant d'emissora de ràdio, amagat rere el títol enigmàtic de *Newton en colors*, el que es presentava com la barreja de moltes cares diferents de les matemàtiques. Almenys així ho vaig veure jo: el punt del tronc on convergeixen

les branques, aparentment sense relació, de la geometria, la topologia, els nombres complexos, el càlcul infinitesimal, les fractals... tot un garbuix que sorprenentment dona resultats bonics.

És aquesta una de les propietats de les matemàtiques que la fan una ciència tan viva, dinàmica i misteriosa. No se sap mai d'on vindrà el pròxim avenç, i no pots menysprear cap tema, ni tampoc assumir que n'hi ha algun d'aïllat: tot es pot estendre a un altre nivell, només cal imaginació. *"Qui té imaginació, que fàcilment treu un món del no-res"* (Gustavo Adolfo Bécquer).

Mais je divague. A més, tenint present la meva motivació principal, vaig veure molt plausible la possibilitat de fer un programa propi amb el qual es fes el "pintat del pla complex" i s'obtinguessin les fractals desitjades (*ad finem*, objectiu complert). I així vaig triar de fer el treball enfocant-lo en aquest tema.

1.3 Objectius

Així doncs, aquest treball té com a un dels objectius apropar tota aquesta informació, que com ja he dit, toca una mica de molts àmbits matemàtics, a un públic no especialitzat. Indiferentment del nivell de coneixements o la relació entre les matemàtiques i l'ofici o els estudis de qui llegeixi aquest treball, el meu objectiu és que es pugui seguir fins al final i gaudir-ne els resultats, ja que no hi ha altra manera: deixar-ho a mitges només farà que tot sembli una pèrdua de temps.

Però alhora, fer-ho sense perdre la formalitat intrínseca que regeix les matemàtiques. Que sigui comprensible no treu que hi hagi nomenclatura, fórmules i simbologia pròpies dels camps estudiats. El rigor és el que separa la ciència de la ficció.

A part d'aquesta filosofia de treball, l'objectiu principal (més relacionat amb el títol del treball) seria tractar el mètode de Newton, comprendre'n el funcionament als complexos i aconseguir fer un programa informàtic que en representi els fractals, utilitzant el llenguatge de programació *C*.

1.4 Hipòtesi

La hipòtesi d'aquest treball és que iterar el mètode de Newton als complexos dona lloc a estructures fractals, relacionades amb els conjunts de Julia, i que és possible fer un programa per a dibuixar aquestes estructures.

1.5 Estructura del treball

Per a arribar al *magnum propositum* caldrà, però dividir el treball en dues parts, teòrica i pràctica:

Marc teòric:

- Introducció històrica, per a situar temporalment (punt 2 *Introducció històrica).*

- Assimilar i definir conceptes per a assegurar coneixements relacionats amb simbologia, derivades, mètodes iteratius, topologia i fractals, reunits en el punt 3 (*Conceptes previs importants*).

- Explicar el funcionament en reals i complexos del propi mètode, als punts 4 i 6, respectivament (*El mètode als reals* i *El mètode als complexos*).

- Repassar l'aritmètica dels nombres complexos (al punt 5, *Nombres complexos*) per a implementar-la en el programa.

Marc pràctic:

- Fer petits programes per a familiaritzar-se amb el llenguatge (amb què no havia treballat anteriorment) i acabar amb un codi de base per a obtenir fractals a partir de funcions del tipus:

$$p(z) = z(z-1)(z-c)$$

recollits al punt 7 (*Estudi local del mètode*).

- Retocar aquest codi base per a trobar les arrels donat un polinomi al darrer punt 8 (*Procediment invers*).

- A l'*Annex* hi ha més exemples de fractals amb altres tipus de funcions.

2.- INTRODUCCIÓ HISTÒRICA

2.1 Precedents (segle XVI)

El mètode de Newton és un procés que busca aproximacions cada cop més bones per a arrels de polinomis. Abans del naixement de Newton, al segle XVI, es van fer grans avenços en matèria de resolució d'equacions. Ja es coneixia una fórmula general per a resoldre les equacions quadràtiques (que avui dia s'ensenya i es fa memoritzar a secundària), una altra per a resoldre les de 3r grau (trobada per Girolamo Cardano el 1545), que no és tan coneguda:

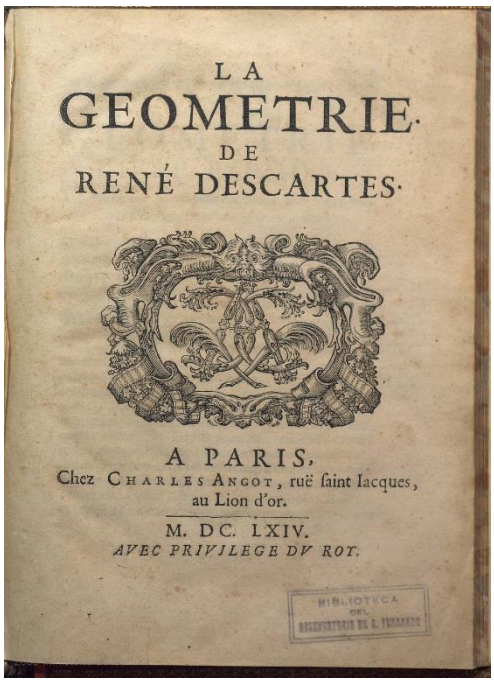
$$x = \sqrt[3]{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)+\sqrt{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)^2+\left(\frac{c}{3a}-\frac{b^2}{9a^2}\right)^3}} + \sqrt[3]{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)-\sqrt{\left(\frac{-b^3}{27a^3}+\frac{bc}{6a^2}-\frac{d}{2a}\right)^2+\left(\frac{c}{3a}-\frac{b^2}{9a^2}\right)^3}} - \frac{b}{3a}$$

Fins i tot per a alguns casos de les de 4t grau (propostes per un deixeble del mateix Cardano, Ludovico Ferrari). Però per a equacions de grau igual o superior a 5 no és possible establir una fórmula general (com no es demostraria fins el 1824, mèrit de Niels Abel).

Però el problema d'aquestes fórmules és que són molt llargues i poc intuïtives. A més, molts cops duien a treballar amb arrels negatives, una feina descrita pel propi Cardano com a "inútil" i "tortura mental". Els nombres imaginaris eren un concepte força impopular.

Per tant calia, o bé desenvolupar un sistema de nombres que permetessin les arrels negatives o bé trobar una fórmula més senzilla per a resoldre els polinomis. Ambdues coses van arribar a mitjan segle XVII, amb els treballs de Newton i Descartes.

2.2 Segle XVII


LA
GEOMETRIE
DE
RENÉ DESCARTES.

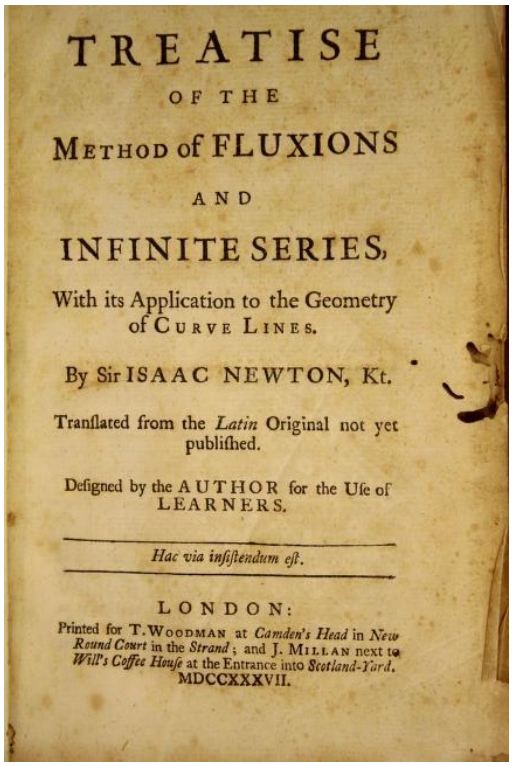
A PARIS,
Chez CHARLES ANGOT, ruë saint Iacques,
au Lion d'or.

M. DC. LXIV.
AVEC PRIVILEGE DV ROY.

A França, el 1637, René Descartes va publicar *La Géométrie* (portada d'una edició de 1664 a la dreta), on va establir una notació per a plans (coordenades cartesianes), i una per als nombres complexos. Tot i que no estava d'acord amb la idea de "imaginari", i opinava que si s'havia de treballar amb aquests nombres, el problema no tenia solució (Newton també opinava així).

Més endavant, el 1665, a Londres va haver-hi un dels darrers brots (i el més mortífer fins aleshores) de pesta que va obligar a confinar tots els estudiants del *Trinity College*, la qual cosa volia dir que Newton havia de retirar-se a Woolsthorpe Manor, la casa que tenia a uns quilòmetres de Cambridge.

Lluny de deixar de banda els estudis, es creu que el geni anglès va aprofitar aquesta situació de tranquil·litat, silenci i aïllament (la qual també ens va tocar viure a mitjan març de 2020) per a investigar sobre les teories més revolucionàries de l'època. Va experimentar amb òptica a la seva habitació, i va prendre moltes



TREATISE
OF THE
METHOD of FLUXIONS
AND
INFINITE SERIES,
With its Application to the Geometry of CURVE LINES.
By Sir ISAAC NEWTON, Kt.
Tranſlated from the *Latin* Original not yet publiſhed.
Deſigned by the AUTHOR for the Uſe of LEARNERS.
Hac via inſiſtendum eſt.
LONDON:
Printed for T. WOODMAN at *Camden's Head* in *New Round Court* in the *Strand*; and J. MILLAN next to *Will's Coffee Houſe* at the Entrance into *Scotland-Yard*.
MDCCXXXVII.

notes sobre càlcul, anàlisi i lleis de moviment planetari (que després van servir per redactar el famós *Principia*). La majoria d'apunts, però, van estar anys amagats a les seves llibretes abans de publicar-los (fet pel qual va tenir problemes amb Gottfried Leibniz per plagi) i no se'n sap la data exacta.

Es coneix que 1667 va poder tornar a la universitat, i al cap de dos anys van nombrar-lo professor. El mètode que duu el seu nom apareix en un dels llibres que va publicar: *Analysis per quantitatum series, fluxiones, ac differentias*, traduït a l'anglès com a *Method of fluxions and infinite series* (portada de la publicació de 1736 a l'esquerra).

Newton, però, en aquest treball matemàtic sobre sèries infinites, vol trobar aproximacions a nombres irracionals (com $\log(1+x)$ o $\sqrt{7}$) utilitzant successions i sumes infinites de polinomis. El mètode que ell detalla és per trobar aquestes successions. El que tracta aquest treball és el que ell mateix descriu com un cas especial del primer, però com una expressió purament algebraica, sense donar cap relació amb el càlcul infinitesimal.

Newton tampoc menciona res sobre "iteració", és a dir, no calculava el terme $\boldsymbol{x_{n+1}}$, sinó que feia tot el procés manual de substitució de termes per a trobar una expressió en forma de polinomi, i després calcular l'aproximació. Més endavant, es va començar a tractar com a mètode iteratiu per a resoldre sistemes no-lineals.

2.3 Segles XVIII i XIX

Els matemàtics van començar a familiaritzar-se amb aquests conceptes gràcies als treballs de Leonhard Euler, el 1777, quan va establir que el símbol $\boldsymbol{i = \sqrt{-1}}$. Poc a poc, la idea de nombres complexos agafava embranzida.

Euler també va tractar una de les identitats considerades més boniques de la història, que incloïa $\boldsymbol{i}$:

$$\boldsymbol{e^{\theta i} = cos(\theta) + i\, sin(\theta)}$$

El 1806, Jean Robert Argand va proposar que es podrien representar en un pla com a coordenades (l'eix X seria per a nombres reals, i l'eix Y per a imaginaris). Carl Friedrich Gauss va popularitzar aquesta idea, i a la representació geomètrica del pla complex se l'anomena *diagrama d'Argand*.

Ell, però, no va ser el primer que va tenir aquesta idea. Contemporani a Newton, John Wallis ja havia escrit sobre aquesta possibilitat el 1685, però va ser ignorat. Aquests nombres van ser acceptats de mica en mica pels matemàtics.

A mitjan segle XIX, també cal destacar els treballs de Georg Cantor, que va co-inventar la teoria de conjunts, i va ser capaç de formalitzar el concepte d'infinit (a partir de conjunts infinits), una idea molt lligada a les fractals i també a les iteracions.

Va fer conegut el *conjunt de Cantor*, obtingut a partir d'extreure la part central d'un segment, i la central dels dos resultants, i així successivament fins a obtenir un conjunt de punts, de mesura nul·la, però alhora infinit. Aquestes idees eren tan revolucionàries per a l'època que a Cantor el van prendre per boig, i fins i tot el van ingressar a clíniques psiquiàtriques.

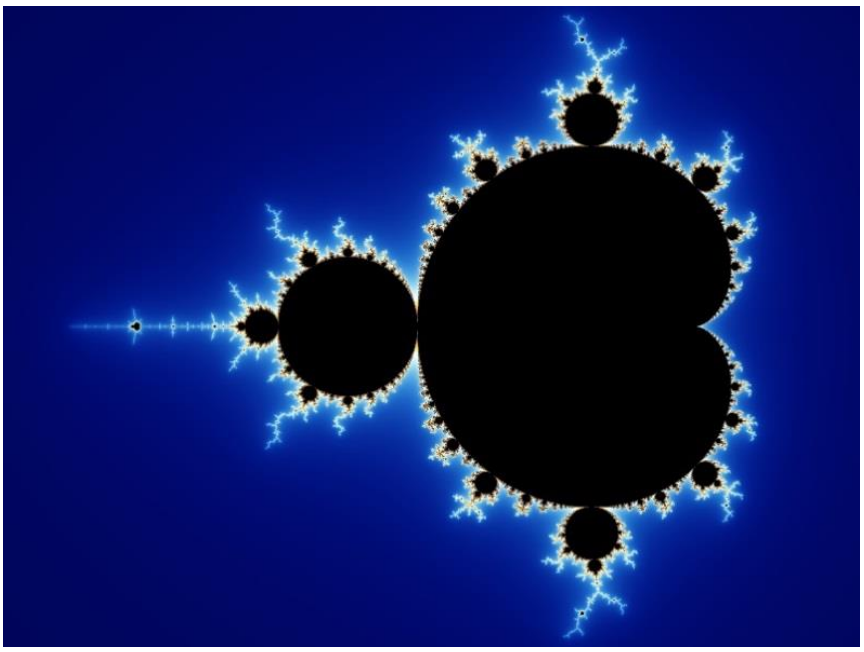
2.4 Segle XX

Tot just començant el darrer quart del mil·lenni, Benoît Mandelbrot va posar per fi nom a aquestes estructures recursives: fractal, del llatí *fractus* (trencat, partit, fraccionari). Tot i que ell no va ser el primer en investigar-ho.

Gaston Julia (matemàtic fill de catalans emigrants) ja els havia estudiat molt abans, però es va veure interromput per la 1a Guerra Mundial, i les seves obres van ser oblidades fins que Mandelbrot les va tornar a citar, apreciant la seva feina. Julia va ser qui va proposar en un article el 1919 que el *conjunt de Julia* (que aleshores rebia un nom diferent) sortís a partir d'iterar funcions en un article que va guanyar molta popularitat.

Aquesta idea de mètodes iteratius, però, la devia prendre de Pierre Fatou, qui va introduir l'anàlisi de funcions per al seu estudi en treballs previs de principi de segle, i que va assentar les bases per al descobriment del *conjunt de Julia*.

Però a diferència de Julia i Fatou, que disposaven d'idees brillants, Mandelbrot va generalitzar els casos donant peu al *conjunt de Mandelbrot* (la imatge fractal més icònica, a baix), va acabar de desenvolupar la matemàtica fractal. Va demostrar que la complexitat visual pot sorgir d'instruccions simples, i que el allò que *a priori* pot semblar un batibull caòtic o un embolic, darrere amaga un ordre subtil regit per normes.



La diferència més considerable és que Mandelbrot va viure suficient com per veure el naixement dels ordinadors, unes màquines que en qüestió de minuts realitzaven les tedioses feines que els seus predecessors havien de fer a base de càlculs sobre paper.

Així, ell i qualsevol persona al món amb un dispositiu amb interfície gràfica tenia l'oportunitat de veure els resultats de les seves investigacions, ja que generaven uns dibuixos (que a mà eren impossibles de fer) amb unes quantes línies de codi.

3.- CONCEPTES PREVIS IMPORTANTS

Per a parlar del mètode de Newton, és convenient fer una base de coneixements que es van repetint al llarg del treball, que ajudaran a entendre-ho millor.

3.1 Simbologia

Per a qui no estigui acostumat, les matemàtiques tenen un conjunt de símbols que s'utilitzen sobretot en les definicions formals. Aquest és un recull de símbols que poden aparèixer en aquest treball:

"$\forall$" per a tot/qualsevol "$\exists$" existeix "$\exists!$" existeix només un cas "$\in$" pertany a

" $\leftrightarrow$ " si, i només si " $\to$ " tendeix a / retorna " $\Rightarrow$ " implica " $\subset$ " conté

" $\cap$ " intersecció de conjunts "$f'(x)$" derivada de la funció $f(x)$ ":" tal/s que

3.2 Derivades

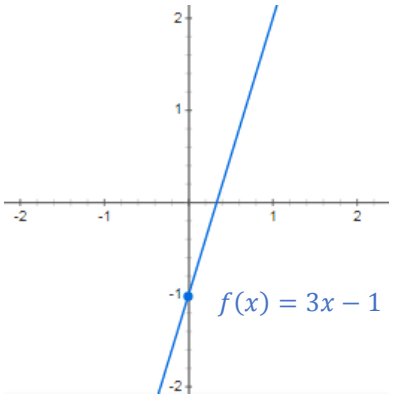
Límits de funcions

Per començar, el tema de les derivades. Abans, però farem un incís en els límits. Concretament en els límits d'una funció. Formalment, el límit es defineix:

$$\lim_{x \to c} f(x) = L \quad \leftrightarrow \quad \forall\, \varepsilon > 0 \quad \exists\, \delta > 0 \quad |x - c| < \delta \quad |f(x) - L| < \varepsilon$$

Vol dir que una funció $f(x)$ qualsevol té límit (de valor L) en un punt c (un valor qualsevol), quan $x \to c$ si i només si, per a qualsevol nombre ε major a zero existeix un nombre δ més gran que zero.

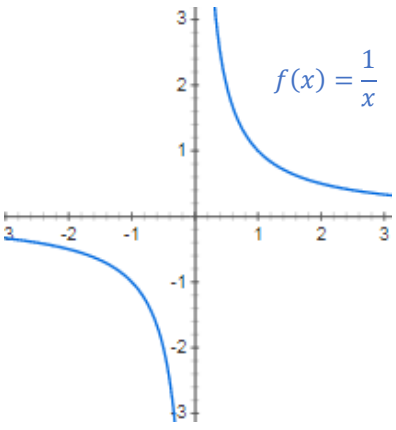
δ representa la distància entre x i el valor c al qual s'està acostant. ε és la distància entre el valor de la funció i el límit. Per tant, si ε i δ són molt propers a zero voldrà dir que la funció sí que té un límit.



Les barres verticals indiquen valor absolut, per tant el límit ha d'existir tant si ens apropem a c des de l'esquerra o des de la dreta. Per exemple, $f(x) = 3x - 1$ (representada al gràfic de la dreta). Si busquem el límit quan $x \to 0$ (amb $\delta = 0.01$) calculem:

$$f(0.01) = 3(0.01) - 1 = -0.97$$
$$f(-0.01) = 3(-0.01) - 1 = -1.03$$

Com que aquesta funció és contínua, el límit és també la imatge del punt en aquest valor. Veiem que s'estabilitzen al voltant de -1 (és a dir, convergeixen), per tant podem dir que $\lim_{x \to 0}(3x - 1) = -1$, tant si ens apropem per la dreta com per l'esquerra i $f(0) = -1$.



Però hi ha cops que no és tan senzill i o bé el límit no existeix o bé la funció és discontínua, i ens trobem amb expressions que no podem avaluar. Per exemple, $\lim_{x \to 0} \frac{1}{x}$. Aquesta expressió (amb $\delta = 0.0001$) dona que:

$$f(0.01) = \frac{1}{0.0001} = 10000$$
$$f(-0.01) = \frac{1}{-0.0001} = -10000$$

En aquest cas, si venim de la dreta (des de valors positius) tendeix a fer-se cada cop més gran, però si venim de l'esquerra (valors negatius), tendeix a $-\infty$. No va cap a cap valor concret: ε és massa gran. Quan passa això es diu que el límit divergeix (o és indeterminat). Hi ha 7 casos d'indeterminacions:

$\frac{k}{0} \Longrightarrow$ Com es veu a l'exemple anterior, pot ser $\pm\infty$ depenent des d'on t'apropis a 0.

$0^0; \infty^0 \Longrightarrow$ Aquí el problema està en què qualsevol nombre elevat a 0 és igual a 1, però 0 elevat a qualsevol nombre és 0. El segon cas es deriva del primer: $\infty^0 = \left(\frac{1}{0}\right)^0 = \frac{1}{0^0}$.

$\frac{\infty}{\infty}; \frac{0}{0}; 0 \cdot \infty; \infty - \infty \Longrightarrow$ Depèn del grau dels polinomis de la funció (normalment es poden simplificar).

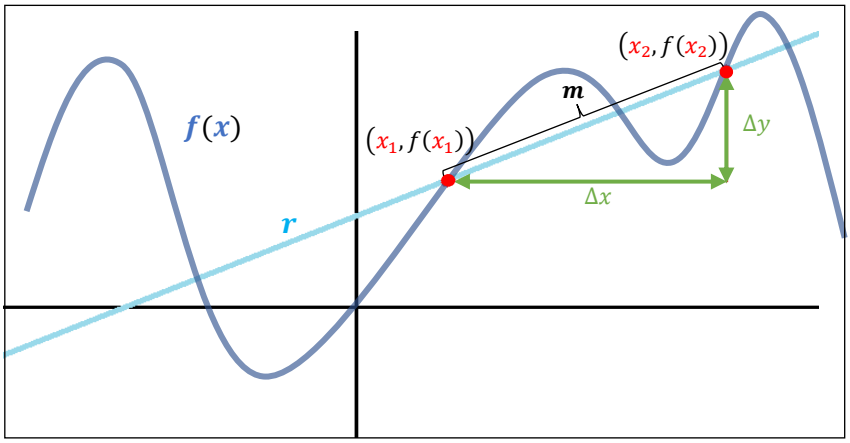
$1^\infty \Longrightarrow$ Si s'arriba a aquest resultat s'han de fer més passos, ja que no és igual a 1.

Derivades

La derivada és el límit de la variació d'una funció $f(x)$ en un punt. La variació d'una funció entre dos punts (o **taxa de variació mitjana**, TVM) és la relació entre l'increment del valor vertical i horitzontal de dos punts qualssevol, $(x_1, f(x_1))$ i $(x_2,\ f(x_2))$:

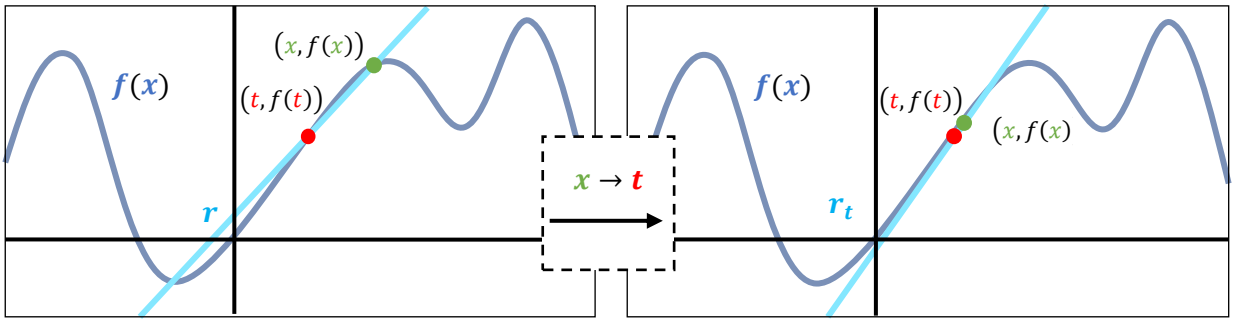
$$TVM = \frac{\Delta y}{\Delta x} = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \boldsymbol{m_r}$$

Aquesta mateixa expressió ens donaria el pendent d'una recta r que tallés aquests punts. El pendent d'una recta (representat amb la lletra $\boldsymbol{m}$) és també la relació entre l'increment del valor vertical i horitzontal de dos punts (una definició anàloga).



Així doncs, la derivada és el límit d'aquesta variació en un punt (t), i definim formalment la derivada:

$$f'(t) = \lim \boldsymbol{TVM} = \boxed{\lim_{x \to t} \frac{f(x) - f(t)}{x - t}} = \boldsymbol{m_{r_t}}$$



Així es veu que en el límit (quan x s'apropa a t), la recta r_t és tangent al punt $(t, f(t))$, i aquesta és la propietat fonamental que aprofitarà el mètode de Newton: la derivada avaluada en un punt retorna el valor del pendent de la recta tangent a aquell punt.

Normes de derivació:

Una cop sabem què vol dir i què implica la derivada d'una funció, hem de saber com derivar-ne. Per fer-ho, se segueixen uns criteris.

La norma general: $(x^a)' = ax^{a-1}$. És a dir, es multiplica la variable per l'exponent i es baixa un grau la funció. Si $f(x) = mx$, llavors: $f'(x) = m$, per definició. La derivada d'una constant és 0, per tant si $f(x) = c,\ f'(x) = 0$.

Hi ha casos especials, però es poden consultar en taules de derivació:

$y = \tan x$	$\begin{cases} y' = 1 + \tan^2 x \\ = \dfrac{1}{\cos^2 x} = \sec^2 x \end{cases}$	$y = a^x$	$y' = a^x \ln a$
$y = \cotan x$	$y' = \dfrac{-1}{\text{sen}^2 x} = -\text{cosec}^2 x$	$y = \ln x$	$y' = \dfrac{1}{x}$
$y = \text{arcsen}\, x$	$y' = \dfrac{1}{\sqrt{1-x^2}}$	$y = \log_a x$	$y' = \dfrac{1}{x \ln a}$
$y = \arccos x$	$y' = \dfrac{-1}{\sqrt{1-x^2}}$	$y = \sqrt{x}$	$y' = \dfrac{1}{2\sqrt{x}}$
		$y = \text{sen}\, x$	$y' = \cos x$
		$y = \cos x$	$y' = -\text{sen}\, x$

Alguns exemples pràctics:

$$f(x) = 2x^4 - x^3 + x^2 - 4 \implies f'(x) = 2 \cdot 4 \cdot x^3 - 3 \cdot x^2 + 2 \cdot x + 0 = 8x^3 - 3x^2 + 2x$$

$$g(x) = \frac{1}{\sqrt{x}} \implies g(x) = x^{-\frac{1}{2}} \implies g'(x) = -\frac{1}{2}x^{-\frac{3}{2}} = \frac{-1}{2\sqrt{x^3}} = \frac{-1}{2 \cdot x \cdot \sqrt{x}} = \frac{-\sqrt{x}}{2x^2}$$

També hi ha moltes funcions que són compostes i es poden aplicar les següents normes.

Suma o resta: la derivada d'una suma o resta de funcions és la suma o resta respectiva de les seves derivades:

$$\big(f(x) \pm g(x)\big)' = f'(x) \pm g'(x)$$

Producte: la derivada del producte de funcions és la suma del producte alternat de derivada i funció,

$$\big(f(x)g(x)\big)' = f'(x)\, g(x) + f(x)\, g'(x)$$

Quocient: Derivada del quocient de funcions:

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)\, g(x) + f(x)\, g'(x)}{g^2(x)}$$

Cadena: la derivada de funcions encadenades és igual a aquesta fórmula:

$$\big(f(g(x))\big)' = f'\big(g(x)\big)\, g'(x)$$

***Regla de l'Hôpital*:** quan tenim una indeterminació del tipus $\frac{\infty}{\infty}$ ó $\frac{0}{0}$, són útils les derivades, perquè són casos en què es pot aplicar la *regla de l'Hôpital*:

$$\boxed{\lim_{x \to h} \frac{f(x)}{g(x)} = \lim_{x \to h} \frac{f'(x)}{g'(x)}}$$

3.3 Mètodes iteratius

Definició i punts notables

Un mètode és iteratiu si, a partir d'un valor, després d'aplicar-li una funció, s'obté el valor següent d'una seqüència:

$$\boxed{A \xrightarrow{\phi(A)} B}$$

És a dir, a partir d'un punt A, aplicant-li una fórmula $\phi(A)$, obtenim un punt B, que serà el pròxim a iterar.

I al valor obtingut també li apliquem aquesta fórmula, i així molts cops (**iteracions**), obtenint una sèrie de valors ordenats, on cada un depèn de l'anterior.

Però evidentment s'ha de començar des d'algun punt. És el que anomenem **condició inicial**, i aquest no depèn de res, sinó que l'escollim. Es representa com a x_0:

$$x_0, \qquad x_1 = \phi(x_0), \qquad x_{n+1} = \phi(x_n)$$

De tota la sèrie de valors que obtenim, a partir d'iterar una x_0, se'n diu **òrbita**:

$$\theta_{(x_0)} = \{x_0,\ \phi(x_0),\ \phi^2(x_0),\ \phi^3(x_0),\ \ldots\ \phi^n(x_0)\}$$

Si l'òrbita es repeteix (retorna la mateixa sèrie de valors de manera cíclica), llavors la condició inicial és un **punt periòdic**, de període p, tal que quan arribem a la iteració p, obtenim el valor inicial:

$$\phi^p(x_0) = x_0 \leftrightarrow \begin{pmatrix} \phi^k(x_0) \neq x_0 \\ \forall\, k \Longrightarrow 1 < k < p \end{pmatrix}$$

És a dir, si $\phi^2(x_0) = x_0$, també $\phi^4(x_0) = x_0$, però el període és 2, no 4.

Si la òrbita té període 1, els punts són **punts fixos**:

$$\phi(P) = P, \qquad \theta(P) = \{P\} \Longrightarrow llavors\ P\ és\ un\ \ punt\ fix$$

Comportament asimptòtic

Els punts fixos poden ser **atractors** o **repulsors**. La definició per a un punt atractor és:

$$P \text{ és atractor} \leftrightarrow \exists\ \varepsilon > 0, \qquad |x - P| < \varepsilon, \qquad \phi^n(x) \to P$$

És a dir, un punt és atractor si les iteracions d'un punt x que és proper a P, tendeixen a P. En canvi, per a un repulsor, encara que comencem a iterar a prop, els valors s'allunyen. Per a definir un punt repulsor, el més senzill és dir que:

$$P \text{ és repulsor} \leftrightarrow P \text{ és atractor en } \phi^{-1}$$

$* \phi^{-1} \neq \frac{1}{\phi}$, sinó que és la funció inversa.

El conjunt de punts que convergeixen cap a un punt atractor rep el nom de **conca d'atracció**. La conca d'atracció (A) d'un punt atractor P, són tots els valors x tals que en iterar-los convergeixen a P:

$$A(P) = \{x\} \quad \Rightarrow \quad \theta(x) \to P$$

Hi ha un teorema que relaciona els punts fixos i el seu comportament asimptòtic. És el ***teorema de l'estabilitat del punt fix***:

$$\phi(P) = P \Rightarrow \begin{cases} si\ |\phi'(P)| < 1 \to P\ atractor \\ si\ |\phi'(P)| > 1 \to P\ repulsor \end{cases} \quad \leftrightarrow \quad \exists\ \phi'$$

Si ϕ és derivable (és a dir, existeix una derivada) i P és un punt fix, en sabrem el comportament segons el valor de la derivada. Si és més petit que 1, atrau; més gran que 1, repel·leix; i si és 1, llavors no ens diu gaire sobre el comportament del punt: hi haurà punts propers que convergiran i n'hi haurà

que no (**comportament inestable**). Com més proper a 0, més atractor. Si és **igual a 0**, parlem de punts **"súper- atractors".**

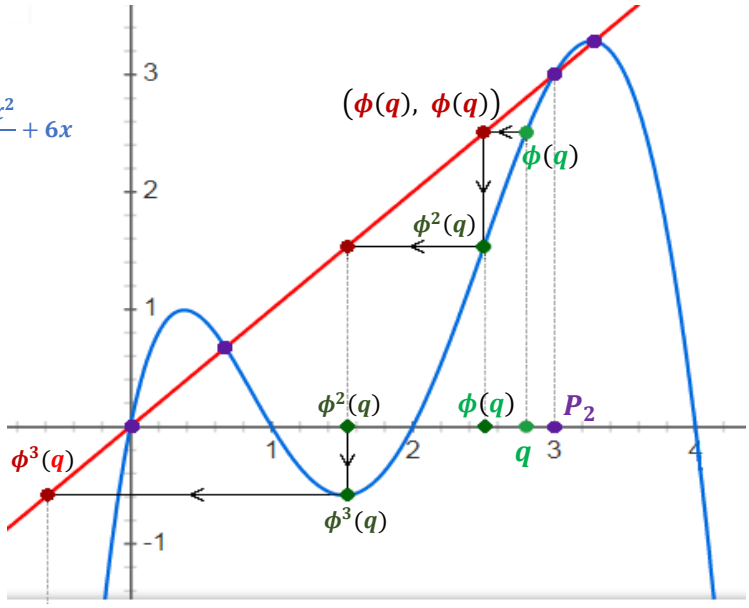
També hi ha una manera gràfica de trobar-los, que consisteix en representar la funció a iterar, $\phi(x)$, la recta $y = x$, i igualar-les. Així, on tallin s'obtenen els punts que compleixen que $\phi(x) = x$: els punts fixos.

Llavors per iterar, comencem a un punt qualsevol q (proper a un punt fix) i fem una línia vertical fins a $\phi(q)$. El següent punt de la iteració, es troba fent una línia horitzontal fins a $y = x$ amb l'objectiu de trobar $(\phi(q),\ \phi(q))$, i una vertical cap avall fins que creui amb la funció $\phi(x)$. En aquesta vertical, el punt que creua amb $\phi(x)$ és $\phi(\phi(q))$, és a dir $\phi^2(q)$.

Llavors, per a tornar a iterar, repetim el procés: tracem una línia horitzontal fins a $y = x$ i en vertical fins a l'eix d'abscisses, i on talla amb $\phi(x)$ obtenim el següent punt.

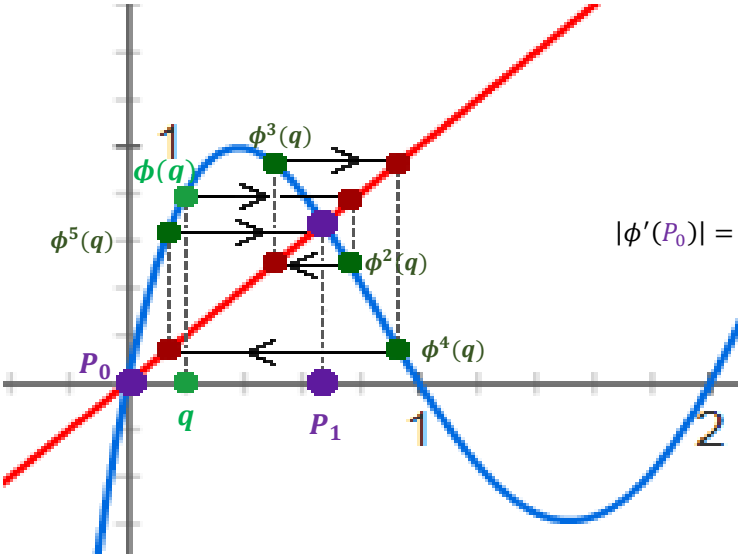
Així, podem veure si el recorregut va cap al punt P_2, aleshores serà atractor o si pel contrari, comença a prop d'un punt fix i marxa a l'infinit, serà repulsor. Si va oscil·lant a esquerra i dreta del punt, P és inestable o és una òrbita periòdica.

$$\phi(x) = \frac{x^5}{12} - \frac{4x^4}{3} + \frac{77x^3}{12} - \frac{67x^2}{6} + 6x$$
$$y = x$$



En aquest exemple, tot i que q és molt proper a P_2, aquest és repulsor, així que:

$$\theta(q) \to -\infty \therefore q \notin A(P_2)$$



En aquest altre exemple, q és més proper a P_0 que a P_1, però, com que $P_0 = 0$:

$$|\phi'(P_0)| = \left|\frac{5\,x^4}{12} - \frac{16\,x^3}{3} + \frac{77\,x^2}{4} - \frac{67\,x}{3} + 6\right| = 6$$

I el teorema del punt fix ens diu que el seu comportament és repulsor, per tant l'òrbita tendeix cap a un altre punt, en aquest cas, P_1.

Tot i això, P_1 tampoc és atractor:

$$P_1 \approx 0.6734\ldots \rightarrow\ |\phi'(P_1)| \approx 1.8530\ldots$$

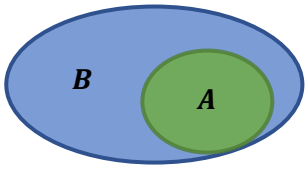
Però és menys repulsor, per tant $\theta(q) \rightarrow P_1$.

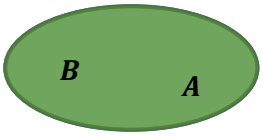
Aquest procediment d'iteració gràfica és poc acurat i pot portar a errors deguts a la precisió, mentre que analitzant les expressions i seguint els teoremes s'obté informació més rigorosa.

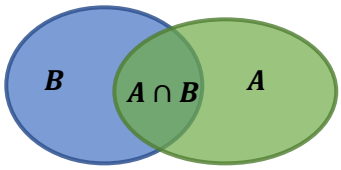
3.4 Topologia

La topologia és la branca de les matemàtiques que estudia el comportament dels espais topològics. Un espai topològic és un conjunt de punts que compleixen unes característiques.

Per exemple, si tenim dos conjunts, A i B:

$A \subset B \leftrightarrow \forall x \in A, x \in B$ 

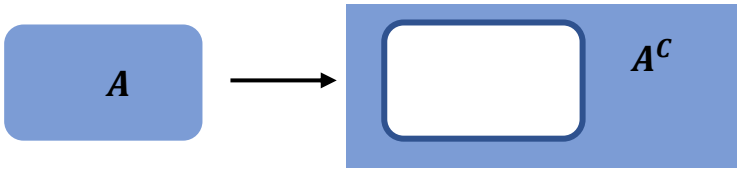
$A = B \leftrightarrow A \subset B, B \subset A$ 

$A \cap B = \{x \in A, x \in B\}$ 

Si agafem un sol conjunt, podem distingir-ne algunes parts característiques:

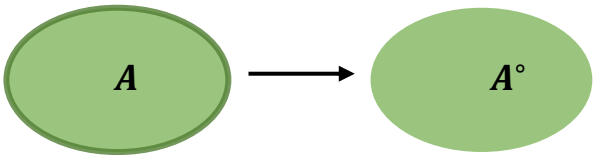
Complementari: $A^C = \{x \notin A\}$

El complementari són tots els punts que no pertanyen al conjunt. Si A^C és obert, llavors A és tancat.



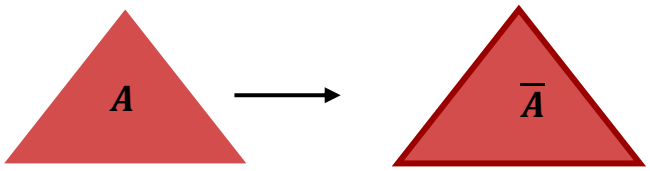
Interior: $\text{int}\,[A] = \{x \Rightarrow \exists\, r > 0 \quad B_r(x) \subset A\}$

L'interior són tots els punts x que estan dins del conjunt, i que per a tots podem fer una bola de radi r centrada a x, i tota la bola també estarà dins el conjunt. Si $A = A°$, el conjunt no té delimitació exterior, i es diu que és obert.



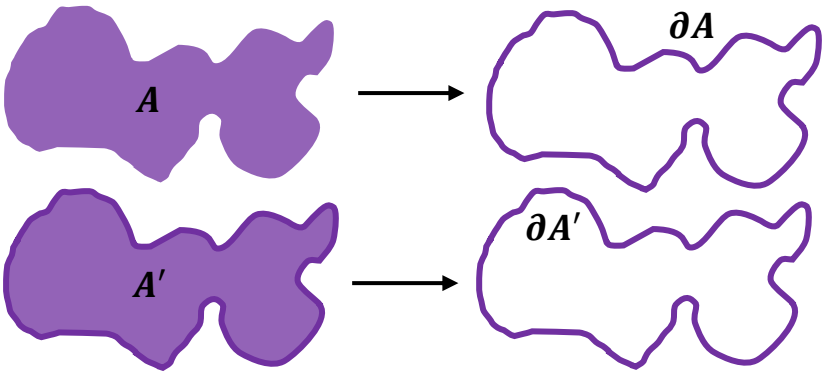
Adherència: $\overline{A} = \{x \in X,\ \forall x_0,\ \exists \varepsilon > 0,\ E = \|x - x_0\| < \varepsilon,\ E \cap A \neq \emptyset\}$

L'adherència són els punts x que, en un entorn E (definit com el voltant de x), la seva intersecció no és un conjunt buit ($\emptyset \not\subset A$). Es relaciona amb el conjunt i l'interior així: $A° \subset A \subset \overline{A}$. Si $A = \overline{A}$; A és tancat.



Frontera: $\partial A = \overline{A} \setminus A^{\circ}$

És a dir, la frontera és l'adherència menys l'interior: el que hem d'afegir a l'interior perquè sigui una figura amb línia exterior. Així ens quedem sempre amb el voltant del conjunt, sigui obert o tancat.



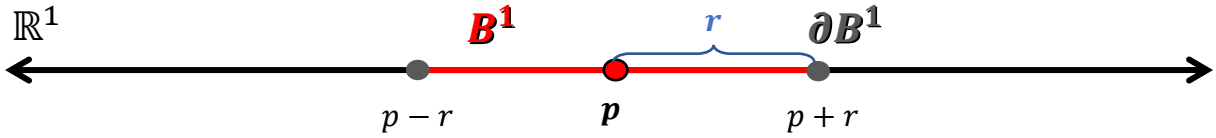
Bola n-dimensional:

Més endavant caldrà saber què és una "bola". La definició formal és la següent:

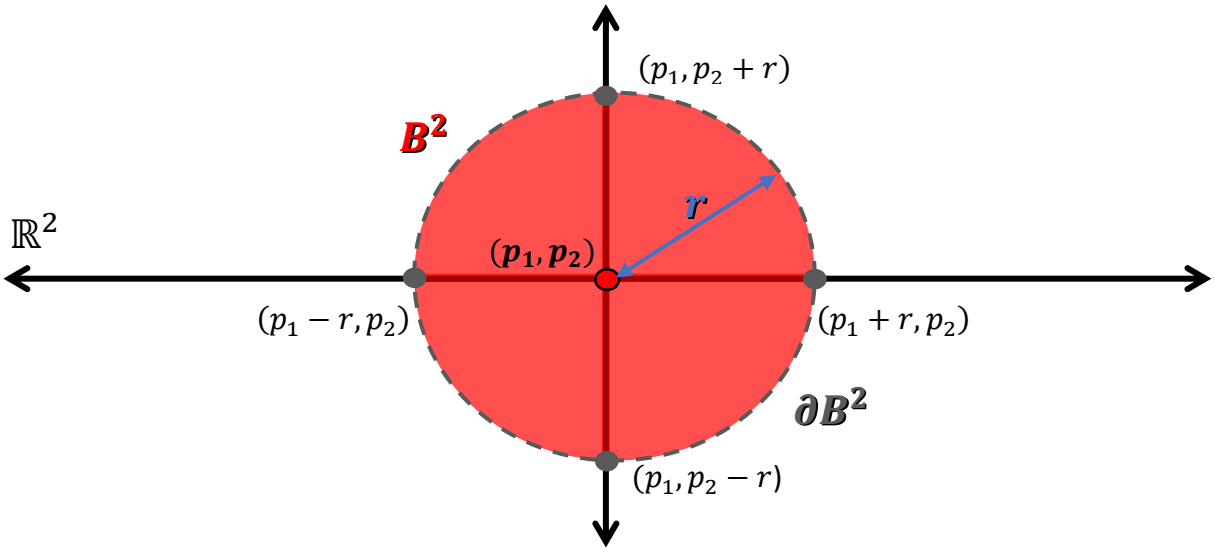
$$B^n(p,r) = \{x \in \mathbb{R}^n : \|x - p\| < r\}$$

Una bola de dimensió n és un conjunt de punts representats sobre $\mathbb{R}^n$ tals que la distància a un punt P, on la bola està centrada, són menors que una distància r. Per tant és un conjunt obert (la frontera no s'inclou) i ple per dins.

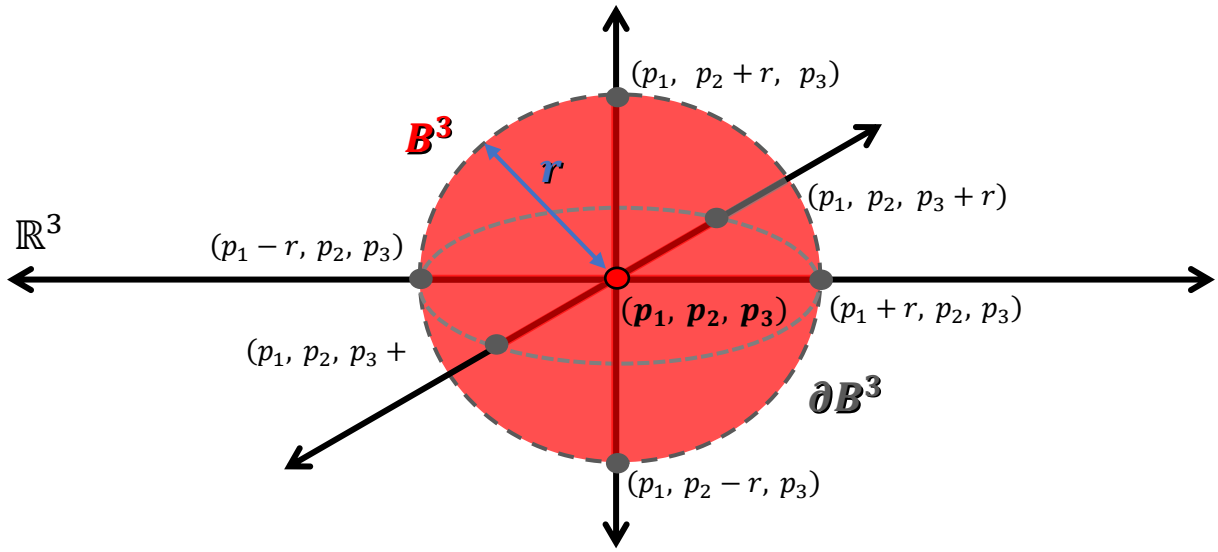
Una bola de dimensió 1, $B^1(p,r)$, correspondria a tots els punts en un **interval obert** de $2r$ amb P al mig. $(P \pm r)$ no formen part de la bola, són la seva frontera: $\partial B^1 = \{p + r,\ p - r\}$.



En 2 dimensions, la bola seria el més semblant a un cercle **sense vora**. El perímetre és ∂B^2, i es veu que $\partial B^1 \subset \partial B^2$.



En $\mathbb{R}^3$, la bola seria una esfera, plena per dins, però **sense contorn**. La superfície de l'esfera és ∂B^3, i $\partial B^2 \subset \partial B^3$.



En general: $\boldsymbol{\partial B^n \subset \partial B^m \leftrightarrow n < m}$, és a dir, si un punt és part de la frontera, en augmentar la dimensió de la bola, segueix formant-ne part (sempre i quan estiguin centrades al mateix punt).

3.5 Fractals

Conjunts fractals

Si apliquem un mètode iteratiu $\phi(z)$, sigui el mètode de Newton o qualsevol altre funció, a tot el pla complex, podem distingir tres espais topològics, segons la convergència dels punts.

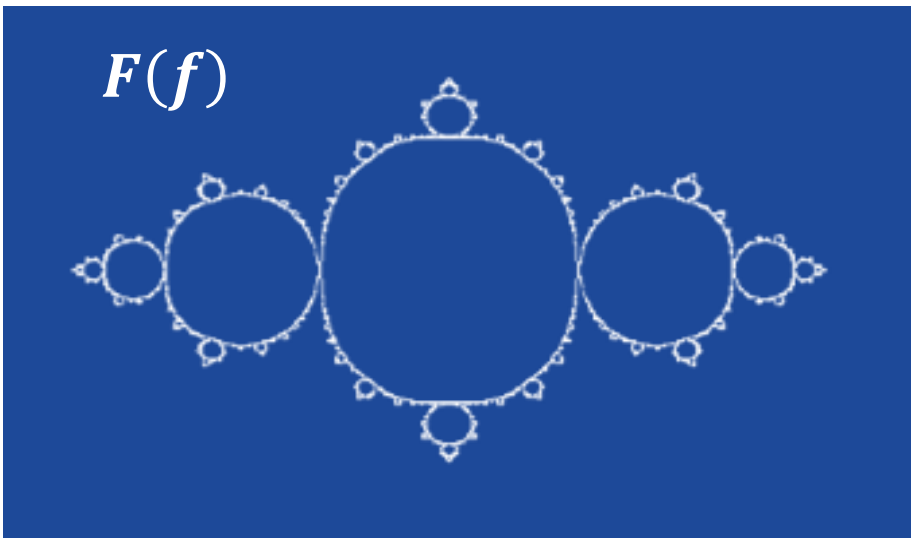
El primer d'ells seria el *conjunt de Fatou* $[F(f)]$, que és el conjunt de punts estables:

$$z \in F(f) \Longrightarrow \exists\, \varepsilon > 0;\quad \forall\, z';\quad \|z - z'\| < \varepsilon : \begin{cases} f^n(z) \to p \\ f^n(z') \to p \end{cases} o \begin{cases} f^n(z) \to \infty \\ f^n(z') \to \infty \end{cases}$$

Si un punt z és estable (pertany al conjunt de Fatou) té, sota iteració, un comportament igual al dels seus veïns: els punts propers, a una distància arbitrària ε, tendeixen a un mateix valor. Aquesta definició també inclou els punts que divergeixen, és a dir: $A(\infty)$.

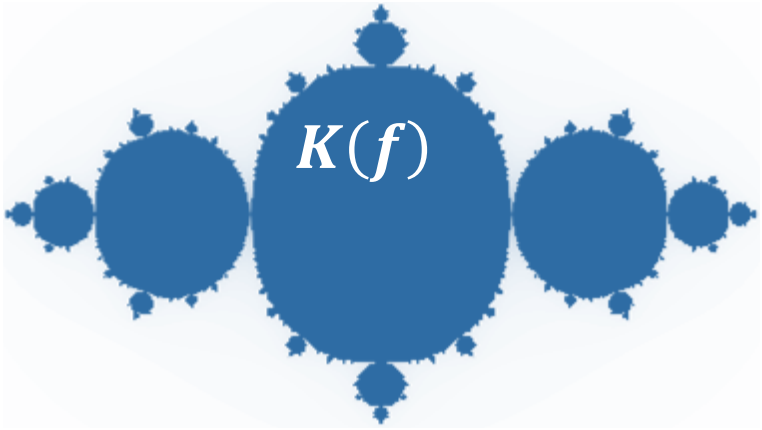
Per tant, els punts que estan a la frontera de dues conques d'atracció **no** compleixen la definició, i no pertanyen a $F(f)$ perquè estan envoltats de punts que tendeixen a nombres diferents, i tenen un comportament inestable.

Un exemple de conjunt de Fatou podria ser el següent[1] (el conjunt és tota la part blava):



També hi ha els *conjunts plens de Julia* $[K(f)]$, que són tots els punts que, a part de ser estables, la seva òrbita no divergeix. És a dir, el complementari de la conca d'atracció de l'infinit:

$$z \in K\,(f);\quad f^n(z) \nrightarrow \infty \quad \Leftrightarrow \quad z \subset A^C\,(\infty)$$



[1] Les imatges d'exemple són resultants d'iterar la funció $f(z) = z^2 - 0.8$

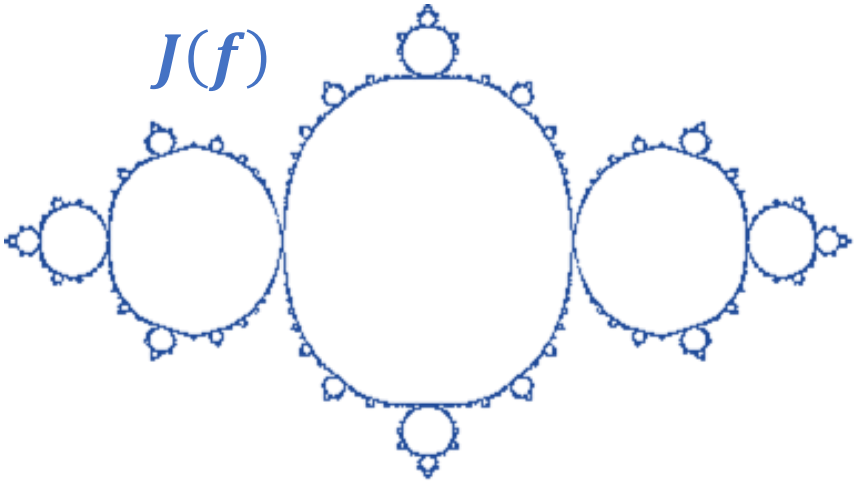
Aquest cop no agafem tots els punts, i es veu que int $[K(f)] \subset F(f)$

Cada conjunt ple, té associat el seu *conjunt de Julia* $[J(f)]$, que és la frontera (no és una figura plena: és la línia que l'envolta) de $K(f)$:

$$J(f) = \partial\, K(f) = \partial\, A^C(\infty)$$

També es pot dir que el conjunt de Julia és el complementari del conjunt de Fatou, ja que comprèn precisament tots els punts inestables:

$$J(f) = F^C(f)$$



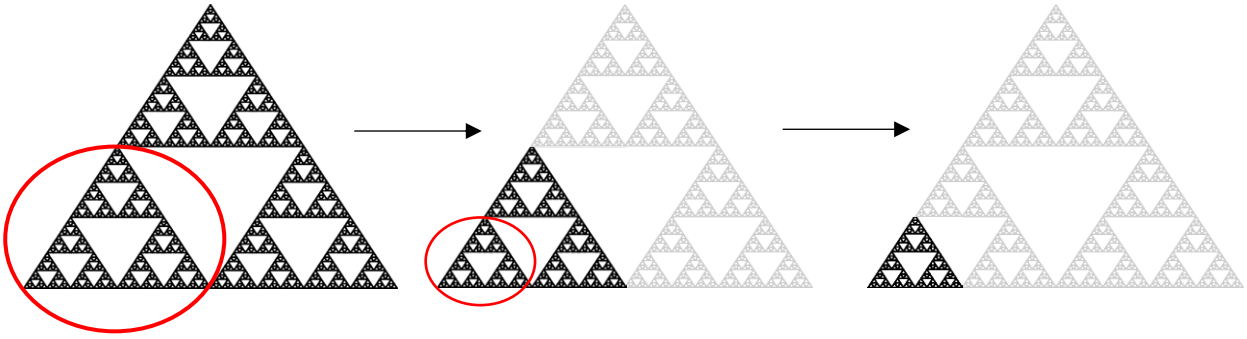
Doncs bé, l'espai topològic que defineix el conjunt de Julia, associat al mètode de Newton, resulta en fractals.

Dimensió fractal

Tot i que s'han intentat donar definicions per a fractal, encara no s'ha arribat a un acord, i no existeix avui dia una única definició. Però n'hi ha dues que són les més comunes[2] :

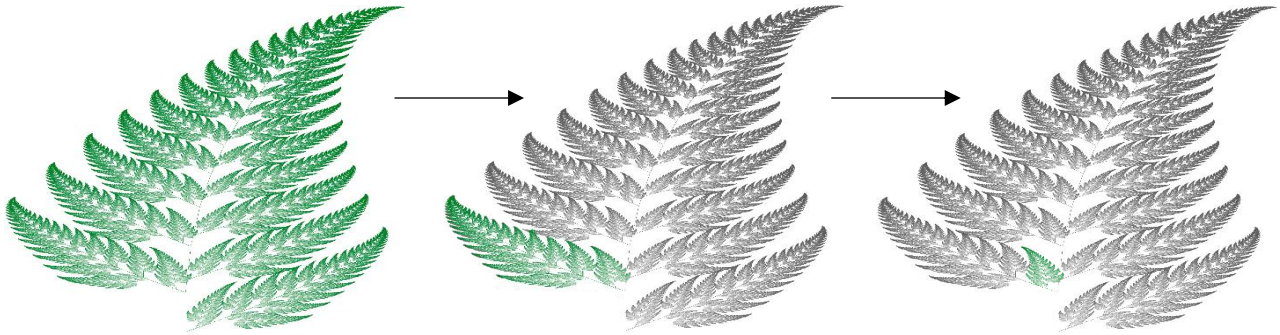
1- Qualitativa: és una definició donada per D. Sullivan, que diu que una fractal és un objecte quasi-auto-similar. Per tant, parlem de similituds en diferents escales, és a dir, apropant-nos a la imatge i donant més quantitat de detalls. Hi ha tres tipus d'autosimilitud, segons la seva exigència:

1. Auto-similitud exacta: l'objecte és idèntic a qualsevol escala (les parts són iguals que el total). Un exemple seria el *triangle de Sierpinski*, en què si agafem un fragment es pot observar que és igual que el triangle sencer, i així successivament:

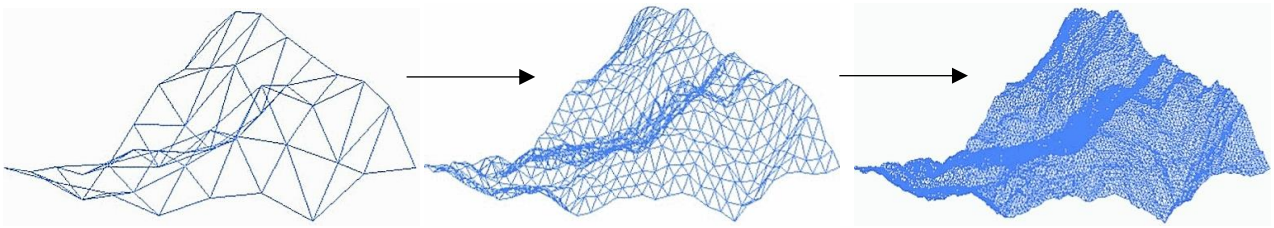


[2] WIKIPEDIA, "*Fractal*" [en línia] < https://es.wikipedia.org/wiki/Fractal#Intentos_de_definici%C3%B3n_rigurosa >

2. Quasi-auto-similitud: les parts són aproximadament idèntiques al total. Poden patir deformacions com rotacions, translacions, estiraments... Es basa en la quasi-isometria. Un exemple seria la *fulla de Barnsley*, una fractal que té parts semblants, però que són una mica diferents degut a la forma de la fractal (estirada i corba) que fa que com més ens apropem a la punta de la fulla, més diferències trobem, però la figura segueix sent la mateixa o molt similar:



3. Auto-similitud estadística: és la que depèn de components aleatoris, però si prenem la fractal com la figura que s'obté en el límit després de que intervinguin tots aquests factors és teòricament auto-similar. Un exemple podria ser la forma que tenen *les muntanyes*. Evidentment no hi ha dues muntanyes idèntiques, però totes s'assemblen molt: els components aleatoris fan que tot i així la forma sigui semblant.

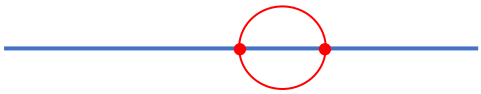


2- Matemàtica: Una definició més rigorosa la va donar Benoît Mandelbrot el 1982, i diu que una fractal és un conjunt en què la seva *dimensió de Haussdorf-Besicovitch* ($\boldsymbol{D_{HB}}$) és estrictament superior a la seva d*imensió topològica* ($\boldsymbol{D_T}$).

La *dimensió topològica* té una definició recursiva. Això vol dir que el següent valor depèn de l'anterior, successivament fins al valor inicial, que hem de definir. Concretament, la D_T és la dimensió de la intersecció del conjunt amb la frontera d'una bola (amb el centre dins el conjunt), sumant 1:

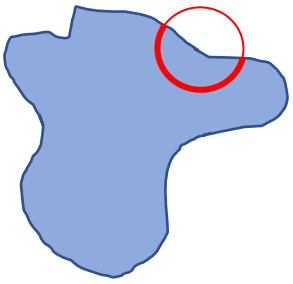
$$\boldsymbol{A \subset \mathbb{R}^n \rightarrow D_T(A) = D_T(A \cap \partial B(p,r)^n) + 1}$$
$$\boldsymbol{\forall\, p \in A;\ \forall\, r > 0 :\ A \cap \partial B(p,r)^n \neq \emptyset}$$

Primer de tot, definim el valor inicial, quan $D_T = 0$. Aquest cas es dona en el **punt**.



Si passem a calcular-ho amb un **segment**, veuríem que la intersecció de la bola són dos **punts** (sigui com sigui el segment), i la seva dimensió seria la $D_{T\ punt} + 1$, és a dir:

$$D_{T\ segment} = 0 + 1 = 1$$



Amb una superfície, la intersecció serà una part de circumferència (un **segment**), per tant una superfície té dimensió topològica:

$$D_{T\ superfície} = 1 + 1 = 2$$

I si ho provéssim amb un **volum**, la intersecció de la bola seria un casquet esfèric, que és una **superfície**, per tant un cos té dimensió topològica

$$D_{T\,volum} = 2 + 1 = 3$$

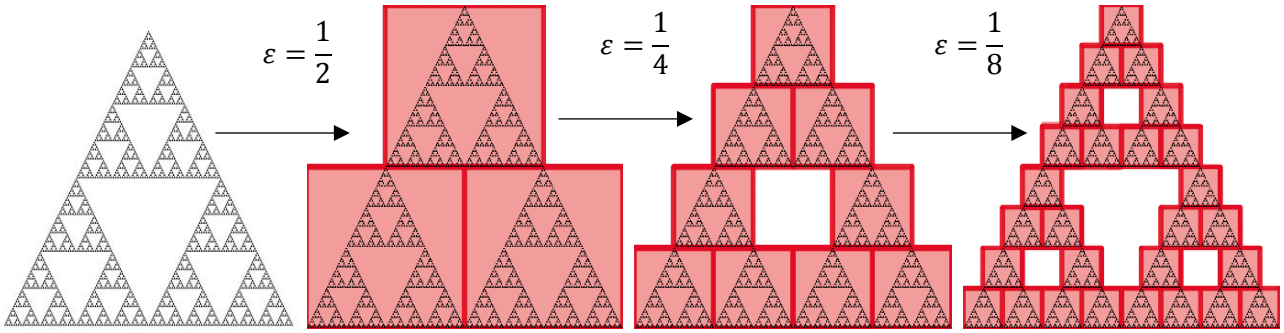
La *desigualtat de Szpilrajn* diu que $D_T \leq D_{HB}$. I Mandelbrot diu que si $D_T = D_{HB}$, la figura **NO** és una fractal, així que per comprovar-ho només hem de trobar D_{HB}. Però per contra, D_{HB} és força més complicada de calcular que la topològica, i requereix de coneixements que no estan a l'abast d'aquest treball.

Tot i així hi ha una altra dimensió que és bastant semblant en valor, que és la de Minkowski-Bouligand ($\boldsymbol{D_{MB}}$). Aquesta es coneix també com *box-counting* (en anglès, *recompte de capses*) ja que consisteix a comptar el nombre mínim de "*capses*" necessàries (de mida de costat de valor $\boldsymbol{\varepsilon}$) que es necessiten per a recobrir completament un conjunt. Aquest *nombre de capses* és la funció $\boldsymbol{N(\varepsilon)}$**.** La dimensió es calcularia:

$$\boldsymbol{D_{MB} = \lim_{\varepsilon \to 0} \frac{\log\left(N(\varepsilon)\right)}{\log\left(\frac{1}{\varepsilon}\right)}}$$

Terme per terme, el logaritme neperià (logaritme en base *e*) del nombre de *capses*, dividit entre el logaritme neperià de la inversa del llarg de les capses (en el límit quan són cada cop més petites).

Per exemple, amb el triangle de Sierpinski, suposant que la base del triangle és de longitud 1 unitat:



Si $\varepsilon = \frac{1}{2}$, calen 3 *capses*; llavors $N(\varepsilon) = 3$.

$$D_{MB} = \frac{\log(3)}{\log(2)} \approx 1{,}5849\ldots$$

Si $\varepsilon = \frac{1}{4}$, en calen 9;

$$D_{MB} = \frac{\log(9)}{\log(4)} \approx 1{,}5849\ldots$$

Si $\varepsilon = \frac{1}{8}$, n'hi ha 27;

$$D_{MB} = \frac{\log(27)}{\log(8)} \approx 1{,}5849\ldots$$

Tots els valors se simplifiquen degut a que són de l'estil:

$$\frac{\log(3^a)}{\log(2^a)}$$

I per propietats dels logaritmes,

$$\log\left(a^b\right) = b \cdot \log(a)$$

$$\Rightarrow \frac{\log(3^a)}{\log(2^a)} = \frac{a\log(3)}{a\log(2)} = \boxed{\frac{\log(3)}{\log(2)}}$$

Per tant el triangle és una fractal, ja que la seva $D_T = 1$ (si fem la prova del cercle, aquest només tocarà en punts) i la seva D_{MB} és aproximadament aquest valor:

$$D_{MB} = \frac{\log(3)}{\log(2)} \approx \boxed{1{,}5849\ldots}$$

Com es veu, és **fraccionària** (no sempre és un enter), cosa que no passa amb la topològica.

Cal aclarir que aquest és un tema d'estudi recent (encara no fa 40 anys de la definició de Mandelbrot), i encara no hi ha consens en una definició complerta i rigorosa que inclogui tots els tipus de fractals.

Segurament no en tindrem cap en els pròxims anys tampoc, ja que la primera definició és poc rigorosa, i la segona exclou fractals en què $D_{HB} = D_T$, que també n'hi ha.

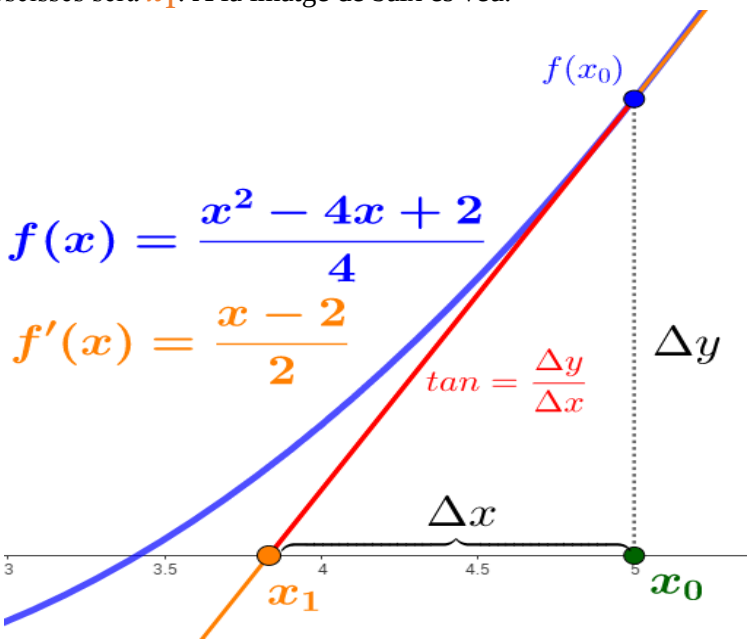
4.- El mètode de Newton als Reals ($\mathbb{R}$)

El mètode de Newton és un mètode iteratiu, la fórmula del qual és:

$$\boxed{N_f(x) = x - \frac{f(x)}{f'(x)}} \Rightarrow \boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$

És a dir, a un punt x_n li restem el quocient entre el valor de la funció i la derivada en x_n.

Gràficament, es tradueix en triar un punt x_0 i fer la recta tangent a $f(x_0)$. El punt on tallen aquesta recta i l'eix de les abscisses serà x_1. A la imatge de baix es veu:



4.1 Deducció de la fórmula

Amb les dades de la imatge, comencem amb x_0 i volem obtenir x_1 per a apropar-nos a l'arrel. Per a fer-ho, hauríem d'operar:

$$x_1 = x_0 - \Delta x \qquad (1)$$

x_0 és coneguda (l'escollim nosaltres com a aproximació), i sabem que el valor de Δx el determina el pendent de la recta tangent a $f(x_0)$:

$$tg = \frac{\Delta y}{\Delta x} \qquad (2)$$

Per tant, per trobar Δx l'aïllem de (2):

$$\Delta x = \frac{\Delta y}{tg} \qquad (3)$$

Però com ja sabem, aquest pendent (tg) és la derivada de la funció en x_0:

$$tg = f'(x_0) \qquad (4)$$

I a més resulta que:

$$\Delta y = f(x_0) \qquad (5)$$

Per tant substituïm (4) i (5) a (3):

$$\Delta \boldsymbol{x} = \frac{\Delta \boldsymbol{y}}{\boldsymbol{tg}} = \frac{\boldsymbol{f(x_0)}}{f'(x_0)} \qquad (6)$$

I de la mateixa manera, (6) a (1):

$$\boldsymbol{x_1} = \boldsymbol{x_0} - \Delta \boldsymbol{x} = \boxed{\boldsymbol{x_0} - \frac{\boldsymbol{f(x_0)}}{f'(x_0)}}$$

D'aquí surt la formula general:

$$\boxed{N_f(x) = x - \frac{f(x)}{f'(x)} \Rightarrow x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}}$$

En l'exemple del gràfic, $\boldsymbol{x_0} = 5$, doncs:

$$x_1 = 5 - \frac{\frac{5^2}{4} - 4 \cdot \frac{5}{4} + \frac{2}{4}}{\frac{5}{2} - \frac{2}{2}} = 5 - \frac{3{,}5}{3} \approx 3.8\overline{3}$$

Amb x_1 repetiríem el mateix procés, i cada cop ens aproparíem més a $f(x) = 0$ (quan creua amb l'eix de les abscisses, és a dir, l'arrel de la funció).

4.2 Punts fixos del mètode

Aquest mètode s'utilitza amb l'objectiu de trobar aproximacions a les arrels d'una funció que s'hi aplica, ja que les arrels són precisament els punts fixos del mètode:

$$\boldsymbol{\alpha}\ \textbf{és punt fix de}\ \boldsymbol{N_f} \leftrightarrow \boldsymbol{\alpha}\ \textbf{és arrel de}\ \boldsymbol{f}$$

Això es pot demostrar de la següent manera:
Si la primera condició és certa, llavors $N_f(\alpha) = \alpha$, per tant, tenim que:

$$N_f(\alpha) = \alpha \Rightarrow \alpha = \alpha - \frac{f(\alpha)}{f'(\alpha)} \Rightarrow \frac{f(\alpha)}{f'(\alpha)} = 0 \Rightarrow \boxed{\boldsymbol{f(\alpha) = 0}} \rightarrow \boldsymbol{\alpha}\ \text{és arrel de}\ f(x)$$

Si la segona condició és certa, llavors $f(\alpha) = 0$, per tant tenim que:

$$f(\alpha) = 0 \Rightarrow N_f(\alpha) = \alpha - \frac{f(\alpha)}{f'(\alpha)} = \alpha - \frac{0}{f'(\alpha)} = \alpha \Rightarrow \boxed{\boldsymbol{N_f(\alpha) = \alpha}} \rightarrow \boldsymbol{\alpha}\ \text{és punt}\ fix$$

Per tant una afirmació implica l'altra. I també podem aplicar el teorema d'estabilitat del punt fix (*veure pàg. 12*). Per a derivar N_f, seguim les normes de derivació (i tenint en compte que $\boldsymbol{f(\alpha) = 0}$):

$$\left|\boldsymbol{N_f'(\alpha)}\right| = \left|1 - \frac{f'(\alpha) \cdot f'(\alpha) - f(\alpha) \cdot f''(\alpha)}{f'(\alpha)^2}\right| = \left|1 - \frac{f'(\alpha) \cdot f'(\alpha)}{f'(\alpha)^2}\right| = |1 - 1| = \boldsymbol{0}$$

Aleshores, el mètode no només funciona, sinó que funciona molt bé, ja que l'arrel és un punt súper-atractor (*veure pàg. 13*), i si arribem a una aproximació bona, la següent iteració no s'allunyarà sinó que serà millor encara: **la convergència és quadràtica**, afegim dos decimals bons a cada iteració. Per això s'utilitza per a trobar les arrels de funcions amb una bona precisió. Això tenint en compte que parlem de **l'estudi local**: si el punt inicial és relativament proper a l'arrel de la funció.

I sempre i quan, $\boldsymbol{f'(x) \neq 0}$, ja que ens trobaríem en un màxim/mínim de la funció, i el pendent de la recta tangent seria 0, per tant seria paral·lela a l'eix d'abscisses i mai tallarien. Aquest punt es podria trobar a la frontera entre dues conques d'atracció o ser una de les arrels de la funció. En el darrer cas, se simplificaria amb el numerador i sí que es podria avaluar l'expressió.

5.- NOMBRES COMPLEXOS ($\mathbb{C}$)

Com que en aquest treball es parlarà del complex, aquest punt servirà per explicar una mica l'aritmètica (com operar) amb els nombres complexos.

5.1 Aritmètica

Suma i resta

Per a sumar i restar, ens convenient tenir els nombres en la forma $\boxed{z = a + bi}$. Aleshores:

$$z_1 + z_2 = (a + bi) + (c + di) = (a + c) + (b + d)i$$

$$Ex: (2 - 5i) + (4 + 3i) = (2 + 4) + (-5 + 3)i = 6 - 2i$$

És a dir, sumant part real i part imaginària per separat. La resta és el mateix, però canviant els signes.

Producte i quocient

Per a multiplicar-los, amb aquesta fórmula s'ha de operar tenint en compte que $\boxed{i \cdot i = -1}$.

$$z_1 \cdot z_2 = (a + bi)(c + di) = (ac - bd) + (ad + bc)i$$

$$Ex: (1 + 2i)(2 - 3i) = \big(1 \cdot 2 - 2 \cdot (-3)\big) + \big((-3) \cdot 1 + 2 \cdot 2\big)i = 8 + i$$

Per a dividir, s'ha de multiplicar pel **conjugat** del denominador als dos termes. Si $z = (a + bi)$, llavors el seu conjugat serà: $\bar{z} = (a - bi)$. Així el denominador és un nombre real, ja que $\boxed{z \cdot \bar{z} = a^2 - b^2}$ (no té part imaginària), i es pot tractar com una multiplicació de nombres complexos.

$$\frac{z_1}{z_2} = \frac{z_1 \cdot \overline{z_2}}{z_2 \cdot \overline{z_2}} = \frac{(a + bi)(c - di)}{(c + di)(c - di)} = \frac{(ac + bd) + (bc - ad)i}{c^2 - d^2}$$

Exponents

Per a treballar amb exponents, va millor utilitzar la **forma polar**, això és expressar el nombre com a producte del seu mòdul i l'angle que forma en el pla complex. Si $z = a + bi$, llavors la forma polar és $\boxed{z = r(\cos\theta + i\sin\theta)}$, on $r = \sqrt{a^2 + b^2}$ i $\theta = \arctan\left(\frac{b}{a}\right)$.

És a dir, r és el mòdul d'un vector que comença a l'origen i acaba a z; i θ és l'angle que forma amb l'eix de les abscisses.

Tot i intervenir la trigonometria, és més fàcil. El que cal saber és que:

$$\cos(\alpha + \beta) = \cos\alpha \cdot \cos\beta - \sin\alpha \cdot \sin\beta$$
$$\sin(\alpha + \beta) = \sin\alpha \cdot \cos\beta + \cos\alpha \cdot \sin\beta$$

Llavors la multiplicació quedaria així:

$$\begin{aligned} z_1 \cdot z_2 &= r_1(\cos\theta_1 + i\sin\theta_1) \cdot r_2(\cos\theta_2 + \sin\theta_2) \\ &= r_1 \cdot r_2\big((\cos\theta_1 \cdot cos\theta_2 - \sin\theta_1 \cdot \sin\theta_2) + i\,(\sin\theta_1 \cdot \cos\theta_2 + \cos\theta_1 \cdot \sin\theta_2)\big) \\ &= r_1 \cdot r_2\big(\cos(\theta_1 + \theta_2) + i\,(\sin(\theta_1 + \theta_2))\big) \end{aligned}$$

És a dir, multiplicar els arguments i sumar els angles. Si $z_1 = z_2$, llavors:

$$z \cdot z = z^2 = r^2(\cos 2\theta + i\sin 2\theta)$$

I per extensió;

$$\boxed{z^n = r^n(\cos(n \cdot \theta) + i\sin(n \cdot \theta))}$$

També canvia una mica el concepte de distància entre dos punts, ja que en estar sobre el pla, pot prendre diferents valors segons el camí per on s'hi arribi.

Allò que en la recta real ($\mathbb{R}^1$) es defineix com el valor absolut de la diferència entre dos nombres ($Dist(r_1 \to r_2) = |r_1 - r_2|$), quan hi ha 2 o més dimensions, hem de parlar de la **norma euclidiana**, que és el mòdul del vector que uneix z_1 amb z_2. Es representa amb dobles barres verticals i es calcula restant les components dels dos nombres i aplicant el teorema de Pitàgores al resultat:

$$Dist(a \to b) = \|a - b\| = \|(a_1 - b_1) + (a_2 - b_2) + (a_3 - b_3) + \cdots + (a_n - b_n)\|$$
$$= \|c_1 + c_2 + c_3 + \cdots + c_n\| = \sqrt{(c_1)^2 + (c_2)^2 + \cdots + (c_n)^2}$$

5.2 Iteració als complexos

També hi ha funcions complexes: $f : \mathbb{C} \to \mathbb{C}$. Són aquelles que treballen amb nombres del pla complex i retornen, després d'operar-hi, nombres del mateix pla. Poden ser idèntiques a les reals, ja que $\mathbb{R}$ és part de $\mathbb{C}$.

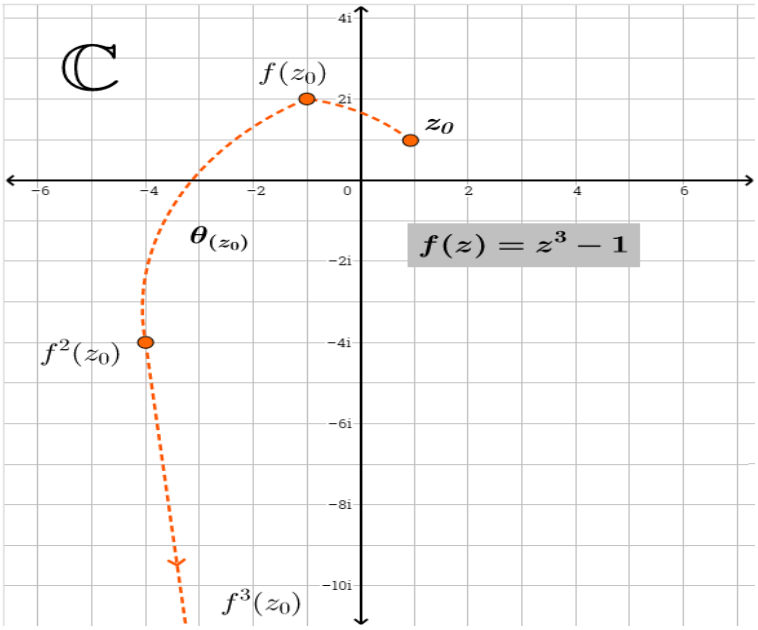
Malauradament, les funcions complexes no es poden representar en un gràfic com les reals, ja que necessitaríem dos eixos per a representar la variable d'entrada i dos més per a la de sortida, un total de 4, i és impossible representar 4 dimensions ($\mathbb{R}^4$).

Tot i així, sí que es poden iterar: $W \xrightarrow{f(W)} Z$, on $W \in \mathbb{C}$ i Z forma part d'una seqüència, i serà el pròxim nombre a iterar (continuaria amb $f(Z)$).

I tenen les mateixes característiques que les reals, com la **condició inicial**, que aquest cop serà un nombre complex amb el qual comencem a iterar: z_0.

Les **òrbites** també existeixen al pla complex, amb la peculiaritat de que ara són les seqüències de punts que segueix l'òrbita, i poden anar en totes les direccions del pla.

$$\theta(z_0) = \{z_0,\ f(z_0),\ f^2(z_0),\ f^3(z_0),\ \ldots\ f^n(z_0),\ \ldots\}$$



Si unim un punt z_0 amb el següent punt de l'òrbita, podem traçar un camí a través del pla. Aquí es veu l'exemple de $f(z) = z^3 - 1$, amb $z_0 = 1 + i$:

En aquest cas, $\theta(z_0) \to \infty$. Però si les òrbites es repeteixen, llavors parlem de **punts periòdics** de període p, que també tenen la mateixa definició que als reals:

$$f^p(z) = z : f^k(z) \neq z$$
$$\forall k : 1 < k < p$$

Un punt z és periòdic i de període p si en la p-ena iteració retorna el mateix punt: $f^p(z) = z$.

Si $p = 1$, parlem de **punts fixos** (complexos):

$$z\ és\ punt\ fix \leftrightarrow f(z) = z$$

Tanmateix, els punts fixos poden ser **atractors** o **repulsors**, i el teorema d'estabilitat del punt fix pels punts del pla complex és molt semblant al dels reals:

$$Si\ \exists f',\qquad f(w) = w \Rightarrow \begin{cases} \|f'(w)\| < 1 \to w\ atractor \\ \|f'(w)\| > 1 \to w\ repulsor \end{cases}$$

Però inclou derivades complexes, i aquest tema no es tracta.

El comportament d'un punt w també es pot definir:

$$w \text{ és atractor} \leftrightarrow \exists\, \varepsilon > 0, \qquad \|z - w\| < \varepsilon, \qquad f^n(z) \to w$$
$$w \text{ és repulsiu} \leftrightarrow w \text{ és atractor en } f^{-1}(z)$$

Només cal tenir en compte que la distància es mesura amb la **norma euclidiana** (*veure pàg. 26*).

Tots els punts atractors tenen les seves respectives conques d'atracció.

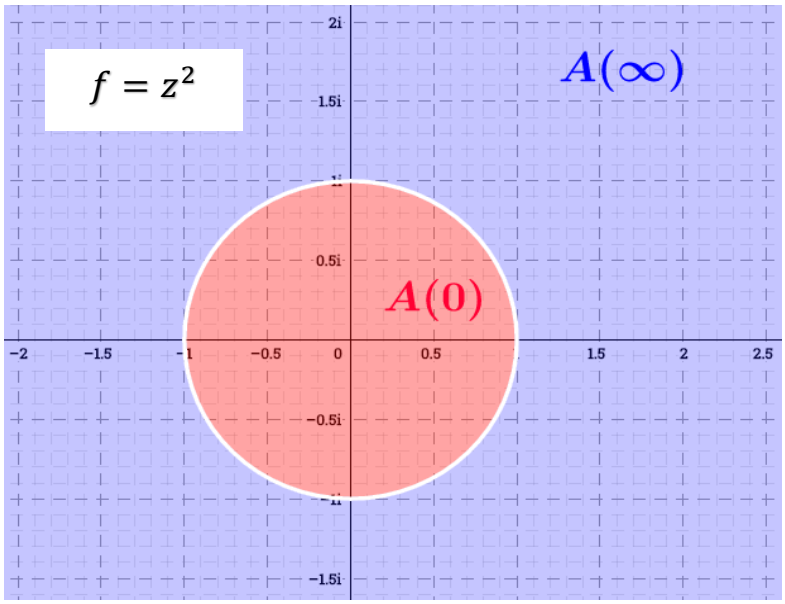
$$z \in A(w) \Longrightarrow f^n(z) \to w$$

Aquestes conques són zones (superfícies) del mateix pla complex, és a dir, hi haurà conjunts de punts que tendiran cap al mateix punt fix, i pertanyeran a la mateixa conca d'atracció.

Per exemple, si $f(z) = z^2 = r^2(\cos(2\theta) + i\sin(2\theta))$. Com que r és el mòdul, tots els punts de dins el cercle unitari (a distància $r < 1$) tendiran a 0, ja que si fas el quadrat d'un nombre més petit que 1 cada cop el resultat és més petit.

Per la mateixa raó, els de fora ($r > 1$) tendiran a ∞, perquè el mòdul s'anirà fent més gran a cada iteració.

El contorn del cercle ($r = 1$) és un conjunt de punts inestables, ja que sempre que els iteres, donen com a resultat un punt del cercle (perquè $1^2 = 1$), però estan envoltats de punts que tendeixen ∞ i punts que ho fan a 0: tenen **comportaments diferents** als dels seus veïns.



Així, el pla queda dividit en dos conjunts separats, Fatou i Júlia:

1. ***Fatou***: conjunt obert de punts estables, amb un comportament igual al dels veïns (correspon a $A(\infty)$ i $A(0)$).

2.1 ***Ple de Julia***: conjunt tancat, de punts inestables, on els veïns tenen comportament diferent (correspon a $A(0)$ i la línia blanca).

2.2 ***Julia***: frontera entre les conques d'atracció (correspon només a la línia blanca).

6.- EL MÈTODE DE NEWTON ALS COMPLEXOS ($\mathbb{C}$)

El mètode de Newton aplicat amb nombres complexos segueix les mateixes pautes que en els reals, però amb les normes d'iteració i aritmètica del pla complex.

Només l'aplicarem amb polinomis, per tant la notació de la fórmula seria la següent:

$$\boxed{N_p(z) = z - \frac{p(z)}{p'(z)}}$$

Tot i que no es pugui representar, i perdi el significat gràfic original, la lògica segueix sent la mateixa. A un punt z del pla complex, apliquem $p(z)$ i $p'(z)$, en fem el quocient i el restem del punt inicial. Repetim aquest procés fins que convergeixi a una de les arrels de $p(z)$.

Només s'han de tenir en compte les normes d'aritmètica dels nombres complexos, però quan es treballa amb ordinadors les operacions són més senzilles de realitzar.

Les derivades complexes no es demostren, ja que la falta d'eines faria perdre rigor a les explicacions, però en el cas dels polinomis segueixen el mateix criteri que les derivades reals, fent servir z en comptes de x com a variable.

El mètode funciona igual de bé. Els punts fixos també seran les arrels de $p(z)$. La demostració és anàloga a la dels reals:

$$\alpha \text{ és punt fix de } N_p \leftrightarrow \alpha \text{ és arrel de } p(z)$$

$$N_p(\alpha) = \alpha \Rightarrow \alpha = \alpha - \frac{p\,(\alpha)}{p'(\alpha)} \Rightarrow \frac{p\,(\alpha)}{p'(\alpha)} = 0 \Rightarrow \boxed{f(\alpha) = 0}\ (\boldsymbol{\alpha} \text{ és arrel de } p(z))$$

$$p(\alpha) = 0 \Rightarrow N_p(\alpha) = \alpha - \frac{p\,(\alpha)}{p'(\alpha)} = \alpha - \frac{0}{p'(\alpha)} = \alpha \Rightarrow \boxed{N_p(\alpha) = \alpha}\ (\boldsymbol{\alpha} \text{ és punt } fix)$$

La prova d'estabilitat del punt fix (*veure pàg. 12*) també és anàloga als reals:

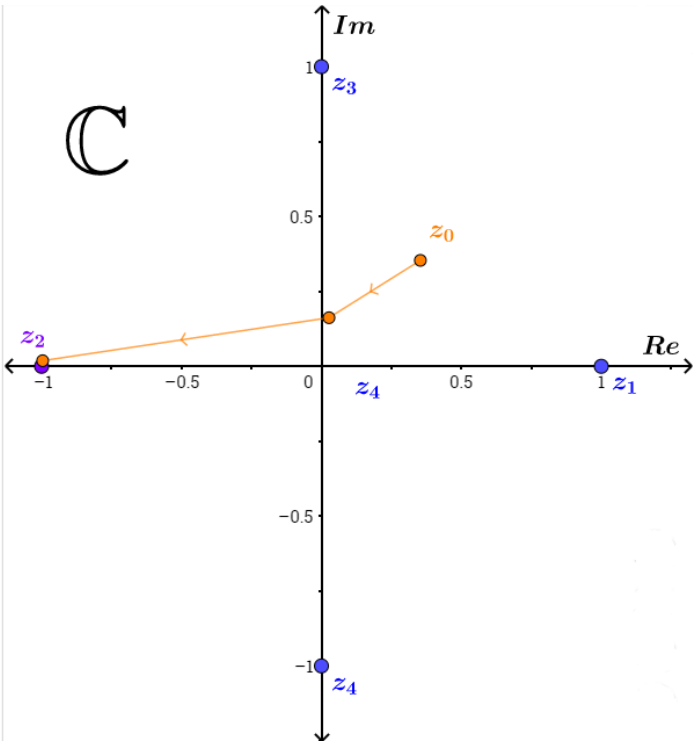
$$\|N_p'(\alpha)\| = \left\|1 - \frac{p'(\alpha)\cdot p'(\alpha) - p(\alpha)\cdot p''(\alpha)}{p'(\alpha)^2}\right\| = \left\|1 - \frac{p'(\alpha)^2}{p'(\alpha)^2}\right\| = \|1-1\| = \|0\| = 0$$

Així que ara els punts fixos súper- atractors (*veure pàg. 13*) són les arrels, tant complexes com reals, i aquest cop poden estar sobre la recta real o a qualsevol lloc del pla.

I també estaran sobre el pla les conques d'atracció: seran àrees (conjunts) de punts en què les seves òrbites convergiran cap a una d'aquestes arrels.

La fractal surt a partir d'aquestes conques d'atracció seguint el següent procediment, que com veureu fer a mà resulta inviable. El més eficient és que un ordinador ho faci amb tots els punts del pla: molt més ràpid i sense errors.

Per exemple, el cas que $p(z) = z^5 + z^4 - z - 1$; les arrels del qual són $z_1 = 1$, $z_2 = -1$, $z_3 = i$, $z_4 = -i$. Llavors els punts fixos (cap on convergiran les òrbites dels punts propers) seran els marcats amb blau:



Llavors, $p'(z) = 5\,z^4 + 4\,z^3 - 1$, i la fórmula iterativa del mètode quedaria:

$$N_p(z) = z - \frac{z^5 + z^4 - z - 1}{5\,z^4 + 4\,z^3 - 1}$$

$$z_0 = 0.5 + 0.5\,i$$

La primera iteració acabarà amb $N_p(z_0) = 0.027 + 0.162\,i$.

Amb la següent ja deduïm cap a quina arrel convergeix: $N_p(z)^2 = -0.996 + 0.019\,i$.

$$N_p(z) \to z_2$$

Podem dir que $z_0 \in A(z_2)$. Aquest camí que està pintat amb fletxes de color taronja és $\theta_{(z_0)}$, no $A(z_2)$.

Però es pot deduir:

$$z_0 \in A(z_2) \Longrightarrow \theta(z_0) \in A(z_2)$$

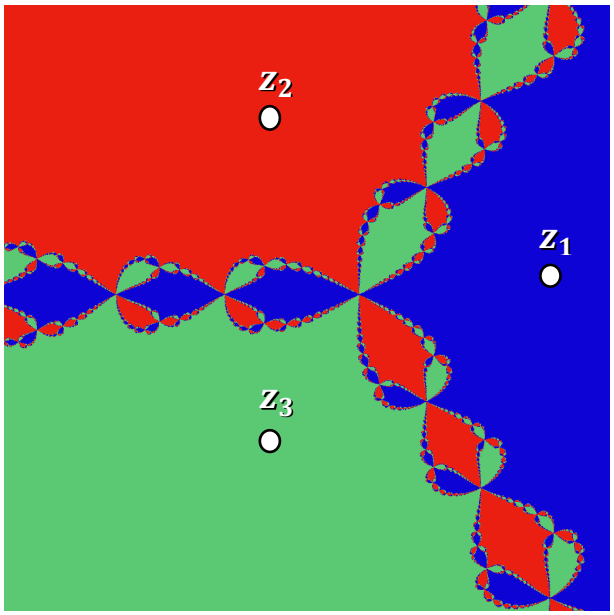
Si una condició inicial pertany a una conca d'atracció, tots els nombres de la seva òrbita també hi pertanyeran.

Per a obtenir la fractal, hem de diferenciar les conques d'atracció, no representar les òrbites. Cal identificar tots els punts d'$A(z_2)$ i marcar-los per distingir-los (pintant-los d'un color determinat).

Es fa el mateix per a les altres tres arrels, fent servir colors diferents per a distingir, i d'aquesta manera quedarien tots els punts del pla pintats de diferents colors (un per a cada arrel).

Per exemple, a la funció $f(z) = z^3 - 1$, on $f'(z) = 3z^2$, la fórmula iterativa del mètode queda així:

$$N_f = z - \frac{z^3 - 1}{3z^2}$$



$f(z)$ té tres arrels, per tant tres colors: $z_1 = 1$, $z_2 \approx -0.5 + 0.866\,i$, $z_3 \approx -0.5 - 0.866\,i$. Segons el color, es distingeixen bé les conques d'atracció.

Ara ens hauríem de fixar en la frontera entre aquestes conques d'atracció. Per definició, és el conjunt de Julia associat al mètode (*veure pàg. 19*):

$$J_{N(p)}$$

Aquesta línia és la veritable fractal, no les zones acolorides!

7.- ESTUDI LOCAL DEL MÈTODE

Per a fer l'estudi local del mètode, el més senzill i eficaç és fer un programa i deixar que l'ordinador s'encarregui de fer els càlculs. A mà seria molt més complicat iterar cada punt del pla. El llenguatge que utilitzem és *C*, i anirem de programes senzills a més complexos, explicant el funcionament de tots.

7.1 "z^2" com a mètode iteratiu

El primer no té a veure amb el mètode de Newton, però sí amb mètodes iteratius. Concretament a z^2, que eleva al quadrat el nombre per obtenir el següent. El què fa el programa és dir si un punt divergeix, $A(\infty)$; convergeix, $A(0)$; o pertany als punts inestables del conjunt caòtic (el cercle unitari).

```
1   /* Titol: z^2 com a metode iteratiu*/
2
3   #include<stdio.h>
4   #include<stdlib.h>
5   #include<math.h>
6
7   typedef struct{                     /* Creem una "estructura" */
8       double a;                       /* que seran els nombres */
9       double b;                       /* complexos amb part real (a) i imaginària (b)*/
10  } complex;
11
```

C és un llenguatge que funciona per llibreries (on hi ha guardades les funcions) i per afegir-les al teu programa cal indicar-ho (`#include<…>`). A continuació definim una estructura (`typedef struct{…} complex;`), amb la qual definim un nombre complex: cada nombre complex té dos arguments, **a** i **b** (ambdós de tipus `double`, que vol dir decimal), que corresponen a la part real i imaginària.

```
12  int main (void){
13      complex z;                      /* Nombre complex "z" */
14
15      double aux;                     /* Variable auialiar per a emmagatxemar z.a */
16      double modul;                   /* Modul de z */
17
18      int itMax = 500;                /* Nombre màxim d'iteracions (no en calen gaire)*/
19
20      printf ("Digues el punt inicial:\n");
21      scanf("%le %le", &z.a, &z.b);   /* Donem valors a z */
22
```

Aleshores, la resta de codi ha d'anar dins de `int main (void){…}`, i dins declarem un nombre **z**, i dues variables decimals, una auxiliar i una altra que serà el mòdul de les iteracions (per a saber si convergeixen o divergeixen). Es declara un enter (`int`) que farem servir de nombre màxim d'iteracions, (aquest programa és senzill i no en calen moltes).

S'imprimeix per pantalla el que hi ha dins de **`printf`**(…), tenint en compte que **"\n"** representa un canvi de línia. Es demanen els valors de **z.a** i **z.b** amb la funció **`scanf`**(…). D'aquesta manera, el nombre complex a iterar queda definit.

```
23      for(int i=0; i<itMax; i++){
24
25          aux=z.a;
26          z.a=aux*aux-z.b*z.b;        /* z^2 = (a+bi)^2 = a^2 + 2abi + (bi)^2 = */
27          z.b=2*aux*z.b;              /*     = (a^2-b^2) + (2ab) i*/
28
29          modul=sqrt(z.a*z.a+z.b*z.b);    /* calculem el mòdul de la z iterada*/
30
31          if(modul>100){              /* Si el modul és molt gran, divergirà */
32              printf("Divergeix");
33              break;
34          }
35
36          else if(modul==1){          /* Si el modul és igual a 1, pertany al cercle unitat *
37              printf("Conjunt caotic");
38              break;
39          }
40
41          else if(modul<0.01){        /* Si el modul és molt petit, convergirà a 0 */
42              printf("Convergeix a 0");
43              break;
44          }
45      }
```

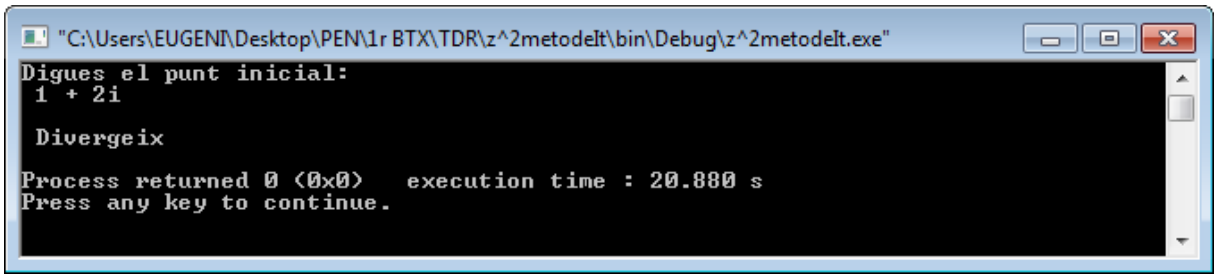
La següent part consisteix en iterar, per a això farem servir la estructura `for(...)` que permet fer una seqüència d'operacions un nombre limitat de vegades, ara **`itMax`**. El que es fa és assignar un nou valor a **`z.a`** i **`z.b`** que és z^2 expressat com `(a+bi)*(a+bi)`, i després es calcula el mòdul d'aquest nou valor.

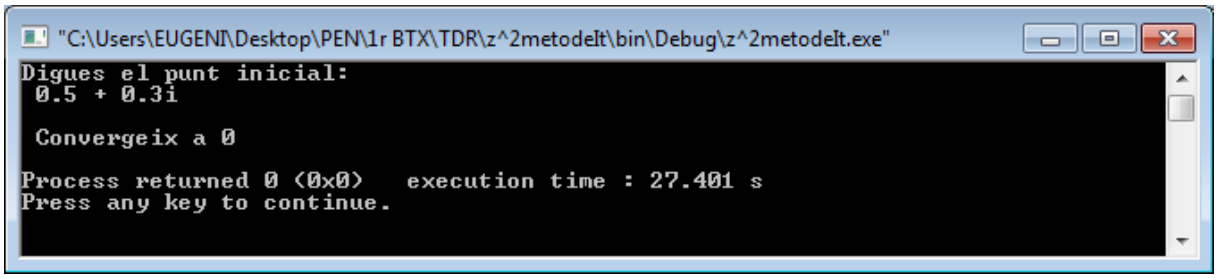
Llavors, si el mòdul és més gran que 100 (`if(modul>100)`), el programa imprimeix per pantalla que el resultat divergeix, i surt del bucle (`break;`). Si el modul és igual a 1 (`else if(modul==1)`), vol dir que ens movem al voltant del cercle unitari, i per tant, el punt pertany al conjunt caòtic. Per contra, si el mòdul és més petit que 0.01 (`else if(modul<0.01)`), el punt tendeix a fer-se més petit cada cop, i convergeix a 0. Els valors de 100 i 0.01 són arbitraris, simplement representen un nombre gran i un nombre petit de referència..

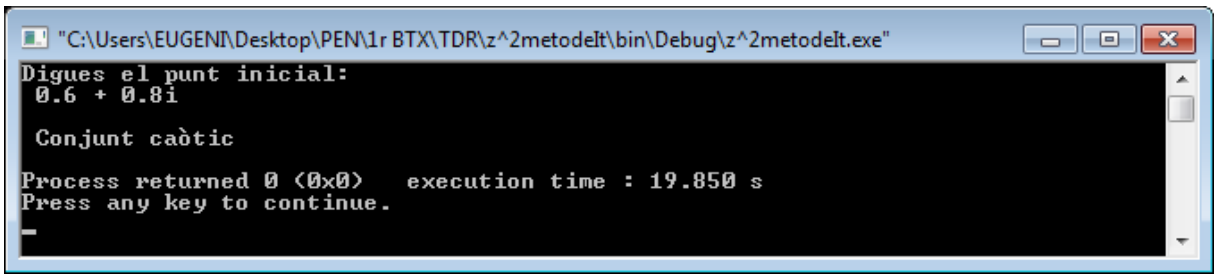
```
46
47
48          return 0;
49      }
50
```

Per indicar que el programa s'ha executat correctament, s'escriu com a darrera línia: `return 0;`.

Iniciem el programa i funciona:







Efectivament, si $z = (1 + 2i)$: $z \in A(\infty)$

Si $z = (0{,}5 + 0{,}3i)$: $z \in A(0)$

I, si $z = (0{,}6 + 0{,}8i)$: z no tendeix a res (es mou al voltant del cercle unitari)

7.2 Iteració del mètode de Newton als complexos

S'aprofiten la inclusió de les llibreries i la definició de nombre complex del programa anterior.

```c
/* Programa per aplicar el mètode de Newton 1 cop a un nombre complex */

#include <stdio.h>
#include <stdlib.h>         // Incloem les mateixes llibreries, i també la de "math.h"
#include <math.h>           // que és la que permet fer operacions com l'arrel quadrada

typedef struct{             // L'estructura de nombre complex és idèntica a abans, però
    double real;            // ara li diem real i imag a la part real i imaginària per
    double imag;            // més claredat a l'hora d'escriure.
} complex;

```

El següent que s'ha de fer és declarar les funcions que utilitzarem, i després, definir-les:

```c
/* Per no omplir el main(), hem de declarar funcions: */
complex suma(complex, complex);     // En el darrer programa no havia calgut, però ara hem de
complex resta(complex, complex);    // declarar funcions. S'escriu el tipus de variable que ha
complex escalar(double, complex);   // de retornar (en aquest cas totes són complexes) seguit
complex mult(complex, complex);     // del nom de la variable (suma, resta, mult...) i entre
complex conjugat(complex);          // parèntesis els arguments, és a dir, el tipus i quantitat
complex divi(complex, complex);     // de variables que necessitem per fer càlculs (en el cas de
complex pol(complex, complex);      // la suma, s'han de sumar 2 complexos i es retorna un nombre
complex der(complex, complex);      // complex, però la norma ha retornar un decimal basat en un
double norma(complex);              // nombre.
```

```c
/* Per definir les funcions, hem d'aplicar
 * el que hem vist a l'aprtat de "Nombres
 * complexos" */

complex suma(complex x, complex y){     /* Per fer la suma:*/
    complex z;
    z.real = x.real+y.real;             // se sumen les parts reals
    z.imag = x.imag+y.imag;             // se sumen les parts imaginàries
    return z;
}

complex resta(complex x, complex y){    /* Per fer la resta:*/
    complex z;
    z.real = x.real-y.real;             // es resten les parts reals
    z.imag = x.imag-y.imag;             // es resten les parts imaginàries
    return z;
}
```

```c
complex mult(complex x, complex y){     /* Per multiplicar:*/
    complex z;
    z.real = x.real*y.real-x.imag*y.imag;   // x*y=(x.real+x.imag)*(y.real+y.imag)=
    z.imag = x.imag*y.real+x.real*y.imag;   // = (x.real*y.real-x.imag*y.imag)+
    return z;                               // = (x.imag*y.real+x.real*y.imag)
}

complex escalar(double a, complex x){   /* Per fer el producte escalar:*/
    complex z;
    z.real = x.real*a;                  // es multiplica part real i part imaginària
    z.imag = x.imag*a;                  // pel nombre que es vulgui
    return z;
}
```

```c
complex conjugat(complex x){            // El conjugat
    complex z;
    z.real = x.real;                    // la part real es queda igual
    z.imag = -x.imag;                   // la part imaginària canvia de signe
    return z;
}

complex divi(complex x, complex y){     // Per dividir:
    complex z;
    z.real = (x.real*y.real+x.imag*y.imag)/(y.real*y.real+y.imag*y.imag);
    z.imag = (x.imag*y.real-x.real*y.imag)/(y.real*y.real+y.imag*y.imag);
    return z;
}
```

Per dividir, es multiplica pel conjugat a numerador i denominador, i el resultat és aquest (expressat en termes de x i y, real $(x.r; y.r)$ i imaginària $(x.i; y.i)$:

$$\frac{x}{y} = \frac{x \cdot \bar{y}}{y \cdot \bar{y}} = \frac{(x.r + x.i)(y.r - y.i)}{(y.r + y.i)(y.r - y.i)} = \frac{(x.r \cdot y.r + x.i \cdot y.i) + (x.i \cdot y.r - x.r \cdot y.i)}{y.r^2 - y.i^2}$$

Els polinomis que iterarem seran de l'estil $p(z) = z \cdot (z-1) \cdot (z-c)$. On c és un nombre complex que podem triar. Així, aquest polinomi té tres arrels: $0, 1$ i c. D'aquesta manera, farem que les conques d'atracció estiguin distribuïdes al llarg del pla.

```
106 complex pol(complex z, complex c){      /* El polinomi (per aplicar el mètode):*/
107     complex x, y, p;
108
109     x.real = z.real-1;      // Serà de l'estil: z*(z-1)*(z-c), on c es pot escollir.
110     x.imag = z.imag;        // Per calcular-lo assignarem a x el valor de (z-1).
111
112     y = resta(z,c);         // I a y, el valor de (z-c)
113
114     p = mult(z, mult(x,y)); // Llavors només cal multiplicar-los tots.
115     return p;
116 }
117
```

També hem de saber el valor de la derivada. La calculem aplicant les normes de derivació:

$$p(z)' = \big(z \cdot (z-1) \cdot (z-c)\big)' = 3\,z^2 - 2\,z\,(c+1) + c$$

```
118 complex der(complex z, complex c){      /* Per aplicar el mètode, cal el valor de la derivada: */
119     complex u, x, D, y;                 // Com que la derivada surt del polinomi d'abans,
120     u.real = 1;                         // seguim les normes de derivació:
121     u.imag = 0;                         // (z*(z-1)*(z-c))' = 3*z*z - 2*z(c+1) + c
122
123     x=escalar(3, mult(z,z));            // assignem aquests valors a x i y
124     y=mult(escalar(2, z), suma(c, u));
125
126     D = suma(resta(x,y), c);            // sumem i ja tenim el valor de la derivada
127     return D;
128 }
129
```

Per saber cap a quina arrel tendeix, assignarem una funció que doni la norma euclidiana d'un nombre complex.

```
130 double norma(complex z){        /* La norma ajudarà a saber a on convergeixen les iteracions:*/
131     double n = sqrt(z.real*z.real+z.imag*z.imag);   // es calcula com ja hem vist anteriorment
132     return n;
133 }
134
```

Ara ja podem començar a escriure la part principal. Declarem z (el nombre a iterar), c (una de les arrels) i u, (una constant que equivaldrà a $1 + 0i$, l'altra arrel). També declarem una variable per portar el compte de les iteracions que portem fins al moment.

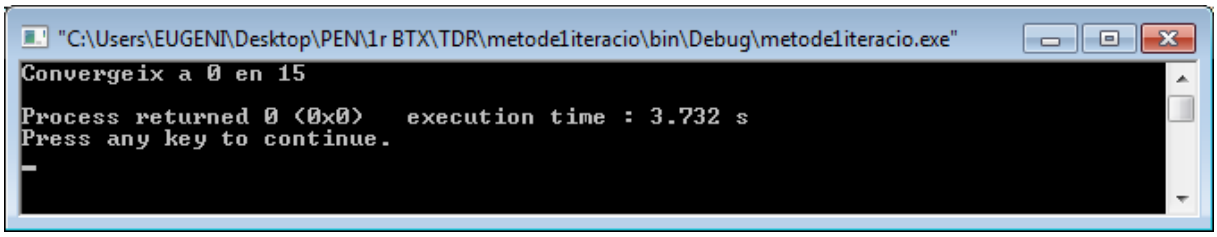
```
23 int main(){
24     complex z, c, u;                 // declararem els nombres que farem servir
25     double epsilon = 1e-4;           // i també una epsilon a mode de tolerància de les iteracions
26
27     z.real = 1;                      // Donem valors a z,
28     z.imag = 2;
29
30     c.real = 0;                      // A c
31     c.imag = 0;
32
33     u.real = 1;                      // I a u, que serà un nombre auxiliar (u = 1 + 0i)
34     u.imag = 0;
35
36     int i;                           // Aquesta variable recomptarà el nombre d'iteracions
37
```

En aquest cas, **`epsilon`** és una variable que servirà de tolerància per a fer els tests de convergència. És arbitràriament petita, i en aquest exemple té el valor de $1 \cdot 10^{-4} = 0{,}0001$.

Com en el programa anterior, calcularem cap a on convergeix, però ara aplicarem el mètode de Newton (línia 39) i després mirarem cap a quina de les tres arrels convergeix. Si arribem a 200 iteracions, voldrà dir que el nombre pertany a la conca d'atració de l'infinit i, per tant, no convergeix.

```
38      for(i =0;i<200;i++){            // Posem un màxim arbitrari, per exemple 200
39          z = resta(z, divi(pol(z,c), der(z,c))); //Per cada iteració, apliquem el mètode.
40
41          if(norma(z)<epsilon){       // Si la distància de z a 0 és molt petita:
42              printf("Convergeix a 0 en %d", i);
43              break;
44          }
45          if(norma(resta(z, u))<epsilon){ // Si la distància de z a u és molt petita:
46              printf("Convergeix a 1 en %d", i);
47              break;
48          }
49          if(norma(resta(z,c))<epsilon){  // Si la distància de z a c és molt petita:
50              printf("Convergeix a c en %d", i);
51              break;
52          }
53      }
54      if(i==200){                     // Si s'arriba al màxim d'iteracions:
55          printf("No convergeix");
56      }
57      return 0;                       // Acabem el programa.
58  }
59
```

Amb els nombres d'exemple, quan $z = 1 + 2i$, i si $c = 0$, $z \to A(0)$ en 15 iteracions donada la tolerància de 0,0001 .



7.3 Mètode i obtenció del pintat del pla

Perquè el programa ens retorni una imatge, hem d'afegir llibreries (que haurien de descarregar-se i col·locar en el mateix directori que el projecte) i paletes de colors (arxius amb extensió *.map*). Així es defineix una funció que apareixerà més endavant. D'altra manera, el programa donaria errors.

```
1   /* Programa per aplicar el mètode de Newton al pla i obtindre un fractal */
2
3   #include <stdio.h>
4   #include <math.h>
5   #include <string.h>
6   #include <stdlib.h>
7   #include <ctype.h>
8
9   #include "image.h"
10
11  typedef struct{             // Mantenim l'estructura de programes anterior
12      double real;
13      double imag;
14  } complex;
15
16  complex suma(complex, complex);      // I les funcions seran idèntiques, ja
17  complex resta(complex, complex);     // el polinomi i la derivada seran del
18  complex escalar(double, complex);    // mateix tipus.
19  complex mult(complex, complex);
20  complex conjugat(complex);
21  complex divi(complex, complex);
22  complex pol(complex, complex);
23  complex der(complex, complex);
24  double norma(complex);
```

La llibreria es diu "`image.h`", la resta és idèntica als programes anteriors. Les funcions tampoc canvien:

```
31  /* Aquí hem d'aplicar el que hem vist a l'aprtat "Nombres complexos" */
32
33  complex suma(complex x, complex y){      /* Per fer la suma:*/
39
40  complex resta(complex x, complex y){     /* Per fer la resta:*/
46
47  complex mult(complex x, complex y){      /* Per multiplicar:*/
53
54  complex escalar(double a, complex x){    /* Per fer el producte escalar:*/
60
61  complex conjugat(complex x){             // El conjugat
67
68  complex divi(complex x, complex y){      // Per dividir:
74
75  complex pol(complex z, complex c){       /* El polinomi (per aplicar el mètode):*/
86
87  complex der(complex z, complex c){       /* Necessitem saber el valor de la derivada per fer el mè
98
99  double norma(complex z){                 /* La norma ajudarà a saber a on convergeixen les iteraci
103
```

El `main()` comença semblant. **`c`** i **`u`** tindran la mateixa utilització que en programes anteriors. La variable **`z`** portarà el recompte dels punts que hem iterat. **`zIt`** serà una variable auxiliar, que serà igual a **`z`** i farem servir per aplicar el mètode de Newton. **`x`** i **`y`** seran enters que farem servir per a editar els píxels de la imatge que volem crear. Com abans, necessitem una tolerància **`epsilon`**.

```
104 int main(){
105     complex z, c, u, zIt;       // Com fins ara, declarem z, c, u i zIt, que serà una
106                                 // variable auxiliar per quan iterem z.
107     int y, x;                   // x, y seran les coordenades dels pixels de la imatge
108     int i;                      // i servirà per iterar el mètode.
109     double epsilon = 1e-8;      // També necessitem la tolerància.
110
111     u.real = 1;
112     u.imag = 0;
113
114     Image *im = createImage(2000, 2000);     // creem una imatge (en blanc) de 2000x2000 pixels
115
116     printf("Escull la part real de c:\n");
117     scanf("%le", &c.real);                   // Inicialitzem "c" amb els valors que es vulgui.
118     printf("\nEscull la part imaginaria de c:\n");
119     scanf("%le", &c.imag);
120
```

Després de declarar les variables numèriques, declarem un altre tipus de variable, que està inclosa en les llibreries: **`Image`**. A aquesta variable li direm **`im`**. Serà un munt de dades que representaran una

imatge de `2000` píxels d'ample i `2000` píxels d'alt. Acte seguit, demanem a l'usuari primer la part real i després la imaginària de **c**.

Amb aquestes dades ja podem començar a iterar el mètode de Newton:

```
121     y=0;        // Per començar, y=0, el primer pixel serà el de dalt a l'esquerra.
122     for(z.imag = 2; z.imag > -2; z.imag-=0.002){
123         /* z.imag anirà de 2 a -2 (de dalt a baix). Cada iteració li restarem 0.002, per
124         tant, al cap de 2000 iteracions haurem acabat. */
125
126         x=0;    // Cada cop que acabem una filera, x=0, tornem al començament de la següent.
127
128         for(z.real= -2; z.real < 2; z.real+=0.002){
129         /* z.real anirà de -2 a 2 (d'esquerra a dreta) i com que z.imag està  fixada, es
130         tradueix en iterar una filera de punts. Cada iteració li restarem 0.002, per tant,
131         al cap de 2000 iteracions haurem acabat. */
132
133             zIt=z;          // La variable a iterar s'actualitza i és el següent punt del pla
134
135             for(i=0; i<50; i++){              /* Iterem el mètode 50 cops */
136                 zIt = resta(zIt, divi(pol(zIt,c), der(zIt,c)));
137             }
```

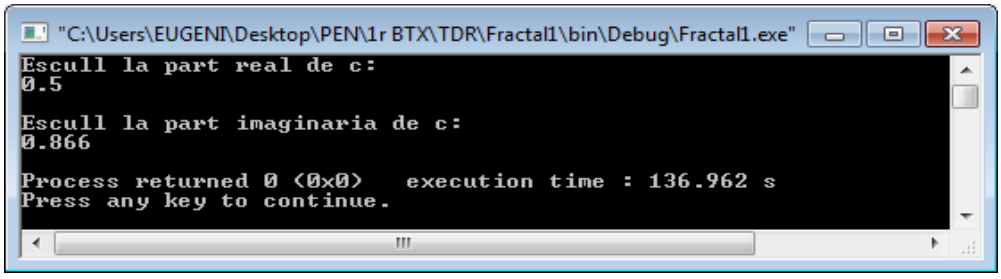
Aquest procediment és molt comú quan es vol cobrir una xarxa de punts: dos `for(...)` un dins l'altre (es diu *nested loops*). El primer cobreix la variable horitzontal, l'altre la vertical. Cada cop que es comença una iteració del primer, es reinicia la variable del segon. Així fem filera per filera, les iteracions de tots els punts d'un quadrant del pla.

Com que la imatge és de 2000x2000 píxels, hem de fer que cada `for(...)` acabi al cap de 2000 iteracions, per tant hem de calcular l'increment (o decrement) de tal manera que no faltin ni sobrin píxels. En aquest cas, $\frac{x_{max}-x_{min}}{2000} = \frac{2-(-2)}{2000} = \frac{4}{2000} = 0{,}002$. Per cada píxel (i punt) apliquem el mètode fins a 50 iteracions (nombre arbitrari, podrien ser més però tardaria més a compilar-se).

```
139             /* fem el test i segons cap on convergeixi donem als pixels diferents colors */
140             if(norma(zIt)<epsilon){
141                 imageSetPixel(im, x, y, 0);   /** tendeix a 0 **/
142             }
143             if(norma(resta(zIt, u))<epsilon){
144                 imageSetPixel(im, x, y, 50);   /** tendeix a 1 **/
145             }
146             if(norma(resta(zIt, c))<epsilon){
147                 imageSetPixel(im, x, y, 100);   /** tendeix a c **/
148             }
149             if(norma(zIt)>4){
150                 imageSetPixel(im, x, y, 255);   /** divergeix **/
151             }
152             x++;    // següent pixel
153         }
154         y++;        // següent filera
155     }
156     imageSaveAsGif(im, "fractal.gif", "bu.map");
157     /* guardem la imatge amb el nom "fractal.gif" i fem servir la paleta "bu.map"
158        que és la que conté els colors verd, blau i vermell. */
159     return 0;
160 }
```

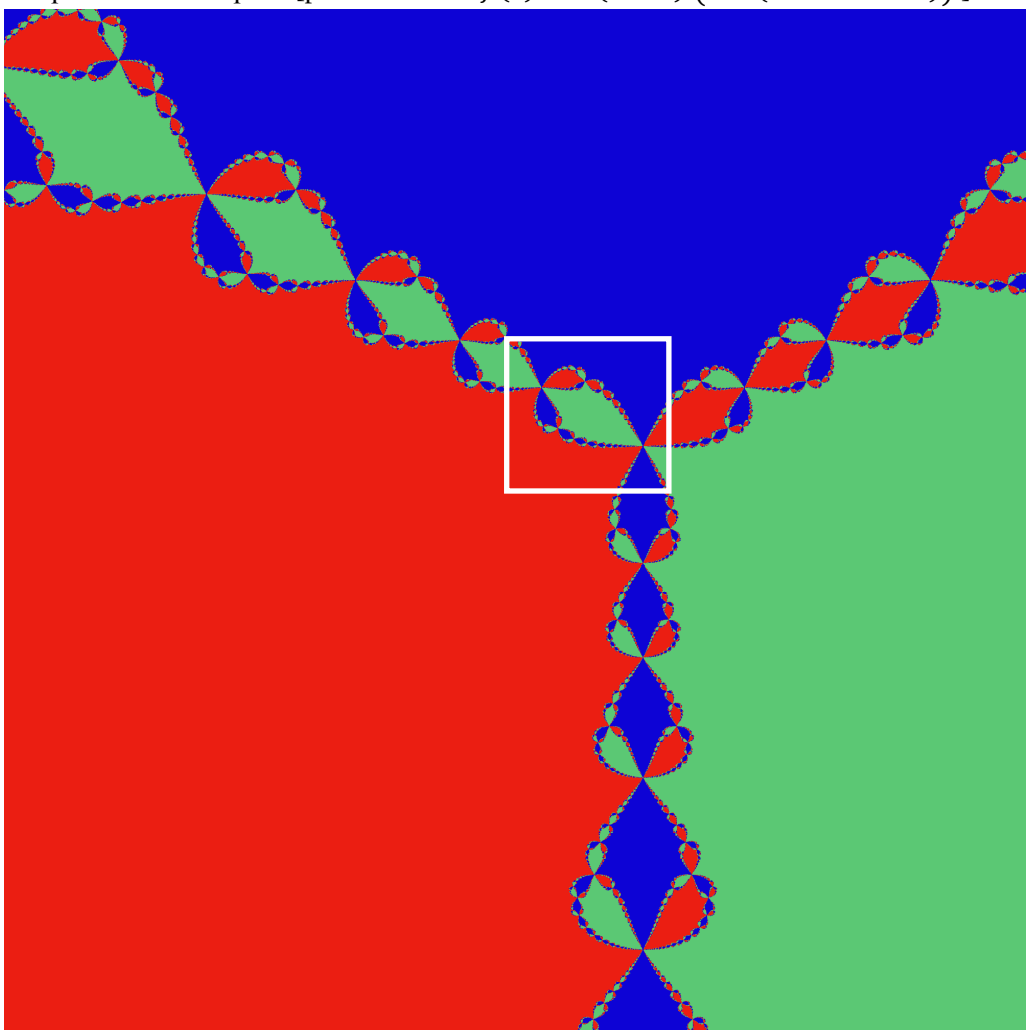
Com en tots els programes, comprovem cap a on tendeix o si divergeix la variable iterada. Aleshores assignem a el punt corresponent (x, y) el color depenent de l'arrel (el 3r argument de la funció **`imageSetPixel(...)`**. Com que només hi ha tres arrels, farem servir `0`, `50` i `100`, que són vermell, verd i blau respectivament.

En el següent exemple, he fet que les tres arrels formin un triangle equilàter, situant c a $(\boldsymbol{cos}(60), \boldsymbol{sin}(60)\boldsymbol{i}) \approx (0.5, 0.866\,\boldsymbol{i})$, d'aquesta manera el fractal serà més simètric:

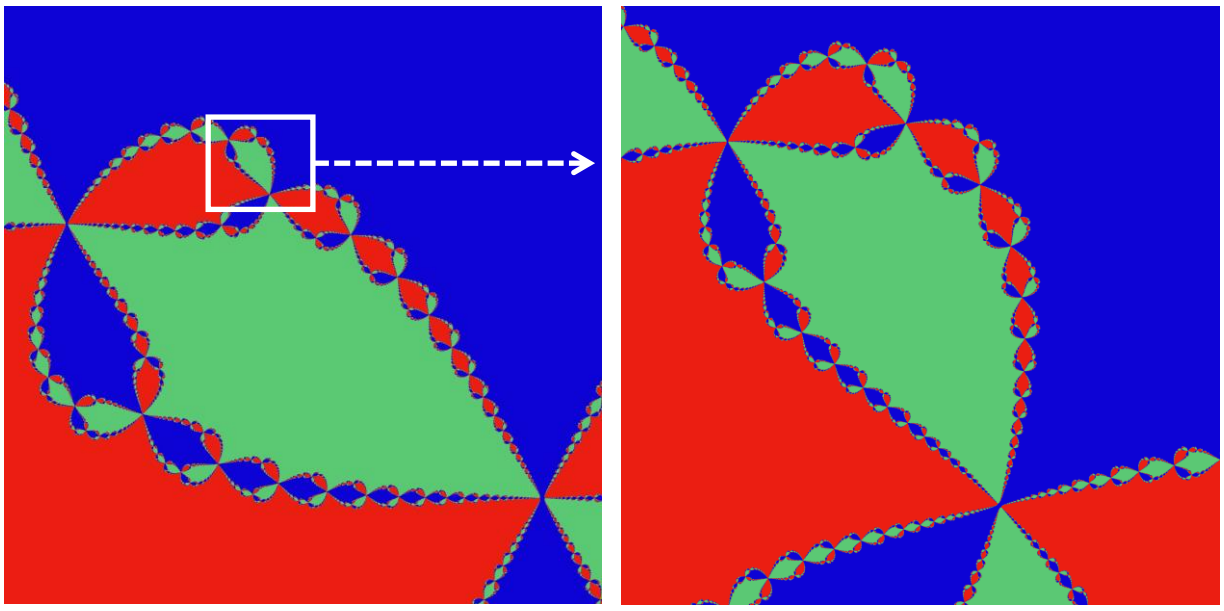


7.4 Resultats (fractals auto-similars)

L'arxiu que es crea és aquest [per a la funció $f(z) = z\,(z-1)\,\big(z-(0.5-0.866\,i)\big)$]:

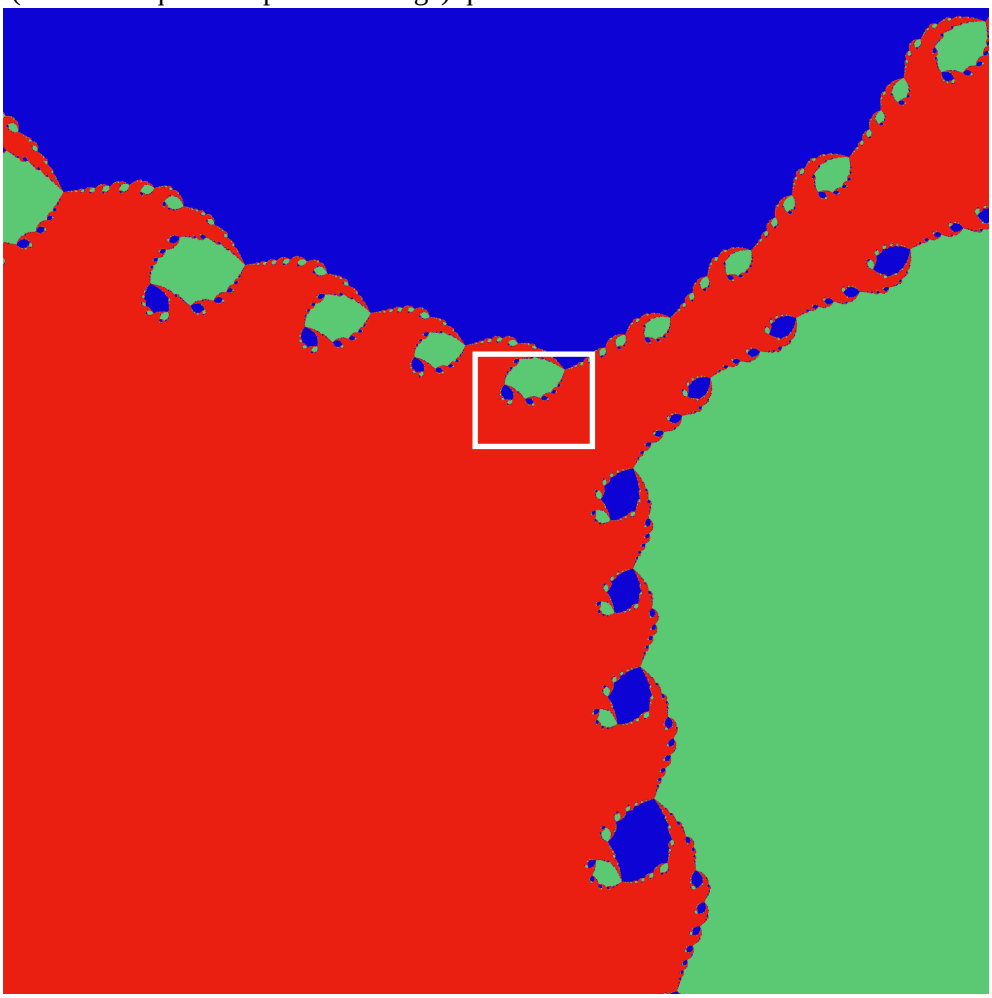


Es pot apreciar la quasi-autosimilitud si ampliem. Per exemple, la imatge de baix a la dreta correspon a la part emmarcada en blanc (de $[0{,}05;\ \ 0{,}55]$ en l'eix real i de $[0{,}2;\ 0{,}7]$ en l'eix imaginari). La de la dreta és una secció ampliada, de $[0{,}225;\ \ 0{,}3]$ en el real i de $[0{,}53;\ \ 0{,}605]$ en l'imaginari:

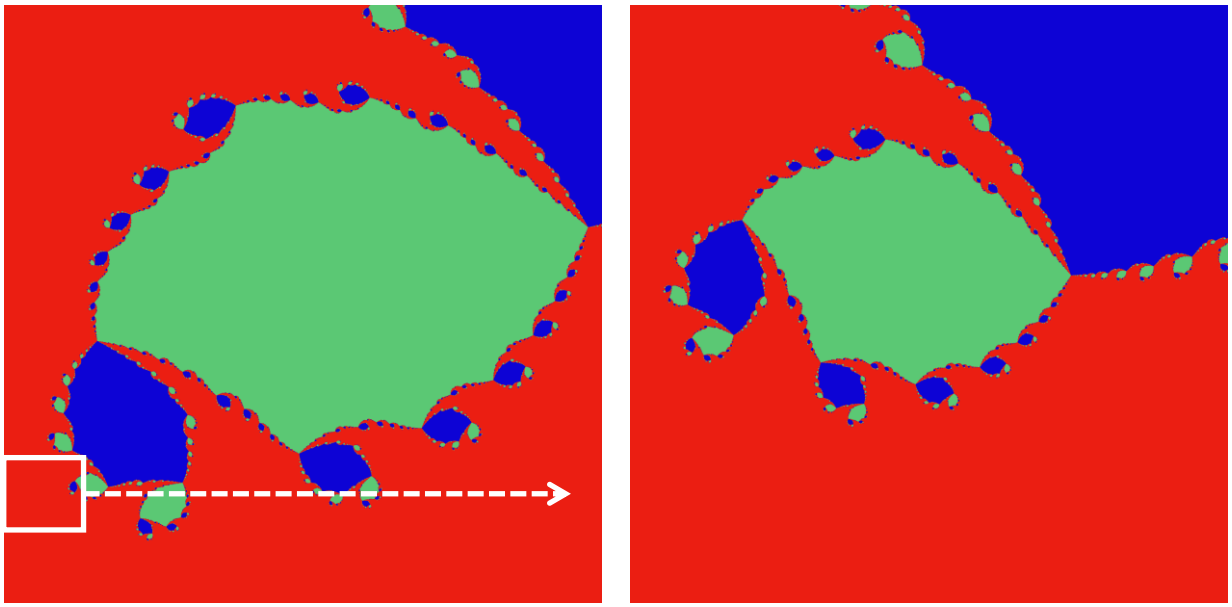


Però què passaria si c no formés el triangle equilàter mencionat abans?

Doncs, per exemple, si $f(z) = z\,(z-1)\,\big(z-(0.25-i)\big)$, la regió de $[-2,\ 2]$ en l'eix real i imaginari (la mateixa que en la primera imatge) quedaria així:



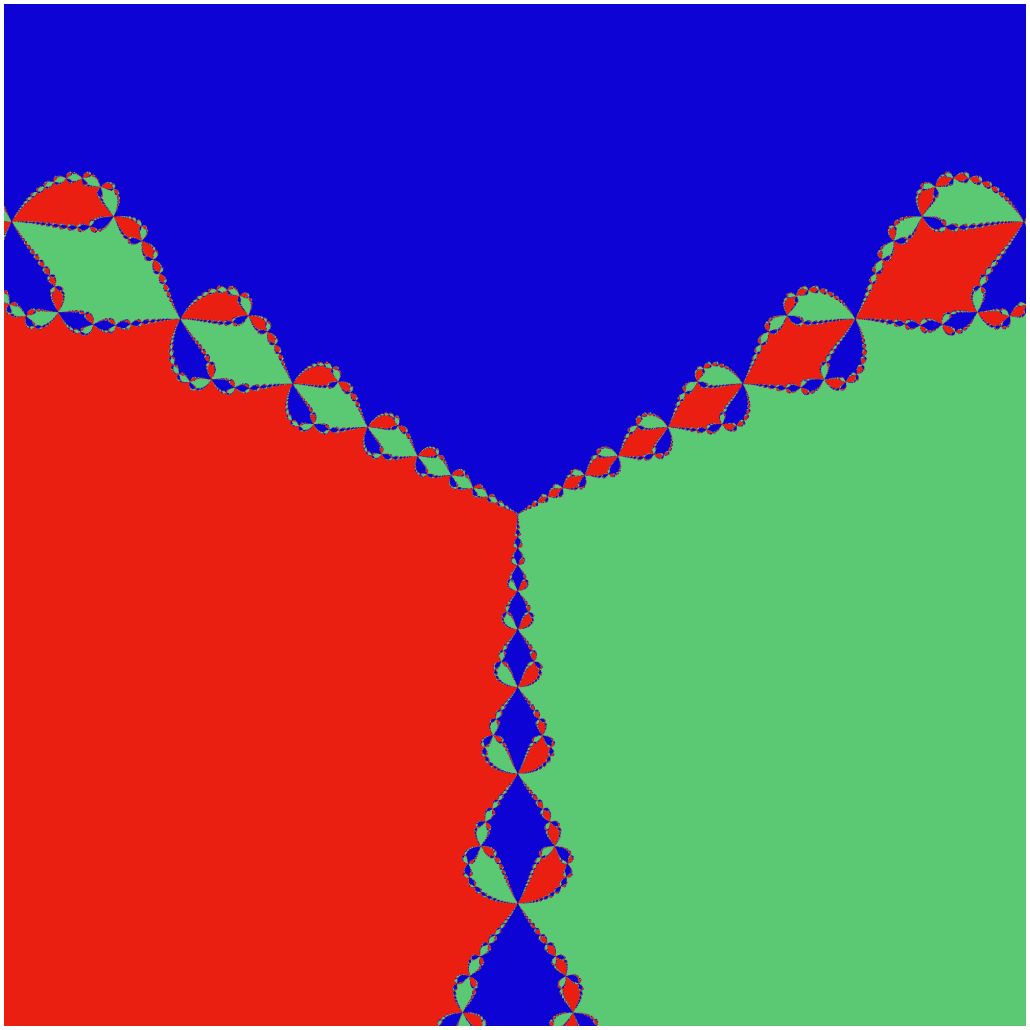
La imatge resultant canvia significativament degut a que les conques d'atracció es mouen i no estan situades en el pla de manera tan simètrica com abans, però segueix presentant quasi-autosimilitud. Aquest cop els intervals són $[-0{,}37 \rightarrow -0{,}12]$ en l'eix real i $[0{,}48 \rightarrow 0{,}73]$ en l'imaginari a l'esquerra, i $[-0{,}34 \rightarrow -0{,}325]$ en el real i $[0{,}51 \rightarrow 0{,}54]$ en l'imaginari a la dreta:



Recalco que la fractal és la frontera d'aquestes figures, **NO** pas els colors. Però d'aquesta manera queda més visual i és més fàcil de diferenciar.

A banda de l'estètica, hem comprovat que, indiferentment de l'escala en què observem, la imatge es va replicant amb petites diferències (quasi-autosimilitud) especialment en el segon exemple, i a mesura que obtenim més i més detall de certes regions, van apareixent formes que recorden al total original. Aquesta és precisament una de les definicions de fractal.

Ara ens podríem preguntar què passaria si, enlloc d'apropar-nos, ens allunyéssim. Doncs la regió de $[-100 \rightarrow 100]$ en els eixos real i imaginari de la primera fractal (quan $c = 0{,}5 + 0{,}866\,i$) queda de la següent manera:



El resultat és poc emocionant, doncs les dues imatges són pràcticament idèntiques. Tampoc són interessants els voltants més pròxims de les arrels del polinomi (que presenten únicament un color) ja que el mètode convergeix molt ràpid degut al comportament atractor.

Les zones que presenten figures més complicades són les que es troben a mig camí de dues o tres arrels: en aquests punts (que estan situats a les fronteres de les conques d'atracció), les òrbites fan salts que poden semblar aleatoris entre les arrels i donen com a resultat al cap de moltes iteracions (després d'estabilitzar el valor) les figures que hem vist fins ara. Aquest comportament caòtic és el que dona lloc a fractals.

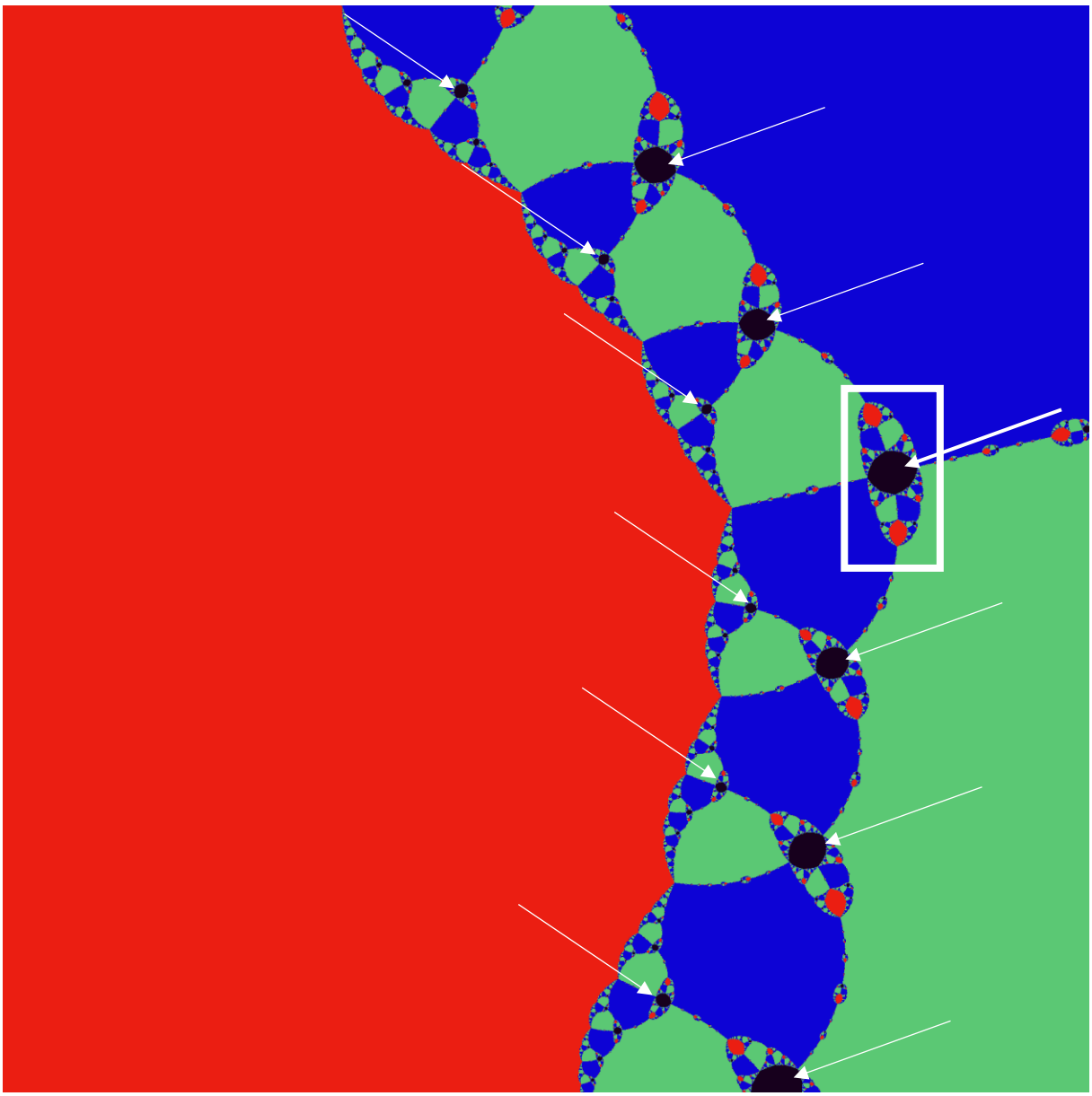
7.5 El mètode falla

En els polinomis d'aquest tipus, hi ha certes "c" amb les quals apareixen zones del pla que acaben de color negre, degut a que acaben convergint a òrbites periòdiques. Per tant, el programa llegeix que no tendeixen a cap de les arrels i compta que divergeixen.

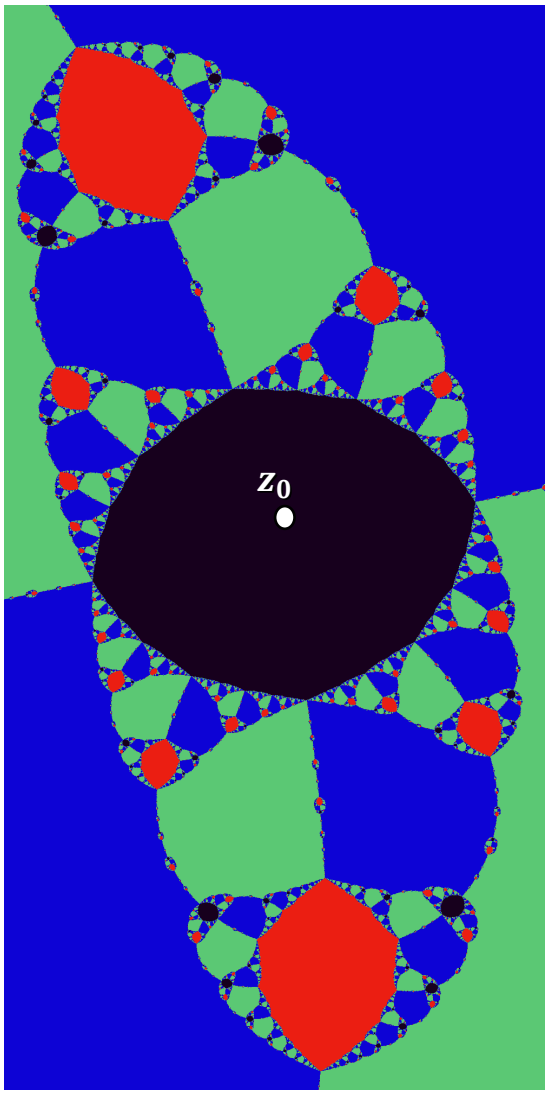
Un exemple d'aquest tipus:

$$f(z) = z\,(z-1)\big(z-(0.905889+0.423516\,i)\big)$$

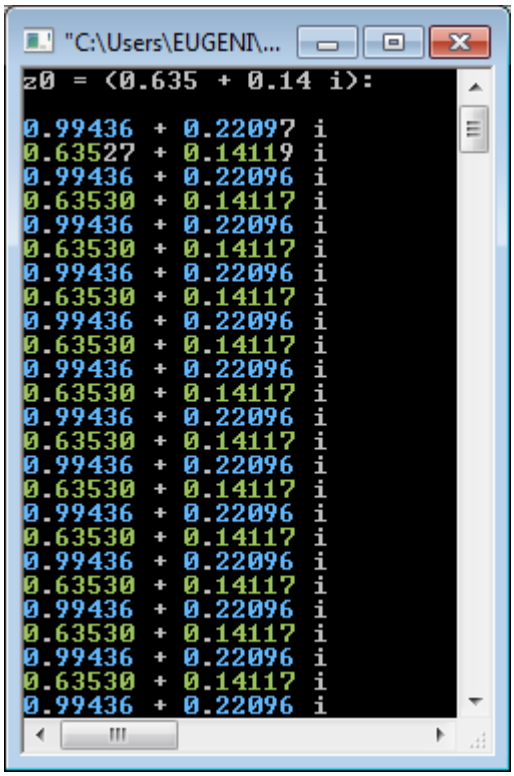
En $\mathrm{Re}[-1;\ 1]$ i $\mathrm{Im}[-1;\ 1]$, les zones fosques són tot de punts que no tendeixen a cap arrel. Algunes les he assenyaladess amb fletxes blanques.



Una ampliació de la regió $\mathrm{Re}[0.57;\ 0.7]$ i $\mathrm{Im}[0;\ 0.28]$ (el rectangle blanc de la imatge de dalt):

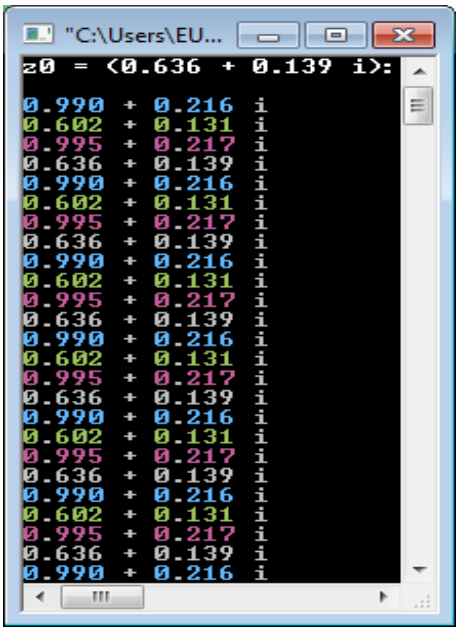


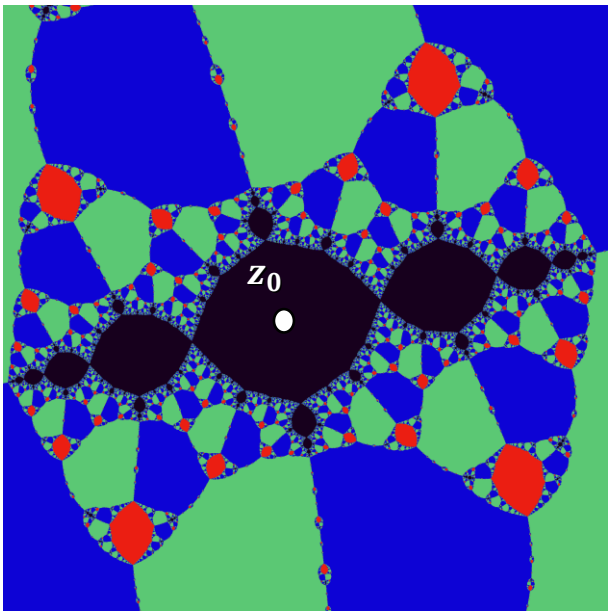
Si comencem a iterar des de dins de la part negra, per exemple, $z_0 = 0.6353 + 0.14117\,i$, les iteracions segueixen el següent patró:



Els parells (**verds**) són iguals, i els senars (**blaus**) també. Per tant, ens trobem en una òrbita de període 2, i els punts van saltant d'un a l'latre, sense deixar que el mètode convergeixi.

Quan $c = 0.90929 + 0.416163\,i$, ens trobem davant d'un cas similar. Aquest cop, però, l'òrbita és de període 4 (cada quatre valors es torna al mateix), i per tant la imatge resultant és diferent. Ampliada a la regió $\mathrm{Re}[0.5725;\ 0.71]$ i $\mathrm{Im}[0.074;\ 0.21]$:





8.- PROCEDIMENT INVERS

Hem vist com obtenir les imatges fractals a partir del mètode. Per fer-ho, necessitàvem conèixer les arrels del polinomi, però també és possible fer el contrari: utilitzar el mètode per a trobar totes les arrels (aproximades) d'un polinomi donat.

8.1 Teorema principal

Comptem amb que tard o d'hora les iteracions del mètode tendiran a una arrel. També hi ha un teorema, proposat a l'article *"How to really find roots of polynomials by Newton's method"* (Hubbard et al., 1998) que diu:

> *"Siguin $\mathcal{P}_d$ tots els polinomis de grau $d \geq 2$, per a qualsevol polinomi $p \in \mathcal{P}_d$ hi ha un conjunt de punts ($S_d = \{x \in \mathbb{C}\}$) compost com a màxim per $N = \lceil 8.32547\ d \log(d) \rceil$ punts en cadascun dels $s = \lceil 0{,}26632 \log(d) \rceil$ cercles amb la propietat que $\exists x \in S_d : x \in A(p_0)\ \forall p_0 \leftrightarrow p(p_0) = 0$."*

Això vol dir que si iterem tots els punts en S_d, segur que obtindrem aproximacions de totes les arrels, ja que no hi ha cap conca d'atracció que no li pertoqui un punt en S_d.

L'operació "$\lceil x \rceil$" es diu *ceiling* i dona com a resultat un enter $k : k \geq x$. Per exemple, $\lceil 2.15 \rceil = 3$; però $\lceil 0 \rceil = 0$.

Aleshores, per un cas general, necessitaríem els N punts que determina el teorema, repartits en s cercles. Per a cada cercle li hem de trobar un radi. Si $1 \leq v \leq s$, obtenim radis en funció de v:

$$r_v = \left(1 + \sqrt{2}\right)\left(\frac{d-1}{d}\right)^{\frac{2v-1}{4s}}$$

I a cada cercle li pertoquen N punts. Si $0 \leq j \leq N-1$, per a cada j obtindrem un angle:

$$\theta_j = \frac{2\pi\, j}{N}, \qquad \Delta\theta = \theta_1 - \theta_0 = \theta_1 = \frac{2\pi}{N}$$

Aquesta fórmula és la que reparteix els punts al voltant de tot el cercle.

Aleshores, un punt qualsevol $x \in S_d$ es pot expressar en forma polar d'aquesta manera:

$$x = r_v \cdot \left[\cos(\theta_j) + i \sin(\theta_j)\right]$$

Exemples per a: $2 \leq d \leq 5$

Grau: d	**Nombre de cercles:** $s = \lceil 0.26632 \log(d) \rceil$	**Nombre de punts per cada cercle:** $N = \lceil 8.32547\ d \log(d) \rceil$	**Radis:** $r_v = \left(1 + \sqrt{2}\right)\left(\frac{d-1}{d}\right)^{\frac{2v-1}{4s}}$	**Increment d'angle:** $\Delta\theta = \frac{2\pi}{N}$
2	$s = \lceil 0.184599 \rceil = 1$	$N = \lceil 11.5416 \rceil = 12$	$r_1 \approx 2.0301$	$\Delta\theta \approx 0.5236$
3	$s = \lceil 0.292582 \rceil = 1$	$N = \lceil 27.4394 \rceil = 28$	$r_1 \approx 2.18149$	$\Delta\theta \approx 0.2244$
4	$s = \lceil 0.369198 \rceil = 1$	$N = \lceil 46.1662 \rceil = 47$	$r_1 \approx 2.24668$	$\Delta\theta \approx 0.1337$
5	$s = \lceil 0.428626 \rceil = 1$	$N = \lceil 66.9966 \rceil = 67$	$r_1 \approx 2.28322$	$\Delta\theta \approx 0.0938$

8.2 Programa per a trobar les arrels

Noves estructures

En aquest nou programa es treballa amb estructures de *punters*, d'*arxiu* i de *caràcters*.

Un **punter** és una variable que pot emmagatzemar un conjunt de valors del mateix tipus. És semblant a una llista de valors. Cada valor té un nombre que l'identifica (l'índex) i s'hi pot accedir individualment. Els farem servir per a guardar tots els valors de les arrels i també els coeficients del polinomi.

La sintaxi és lleugerament diferent que la de les estructures normals. El més important que cal saber és que es declaren amb un asterisc davant del nom (`double *nom_del_punter` és un punter que guarda variables tipus `double`) i que per accedir a un valor es posa el nom de la variable seguit de l'índex (un enter que pot ser `0`) entre claudàtors: `nom_del_punter[index]`.

Una variable **arxiu** (o **FILE**) es fa servir si es vol llegir o escriure un fitxer de l'ordinador. Si es vol escriure en un fitxer amb el nom de la variable, però en el directori no n'hi ha cap, el programa en crea un. La funció per escriure-hi coses: `fprintf(nom_del_fitxer, "coses a escriure-hi");`

Una variable de **caràcters** (o **char**) és senzillament un nom, una cadena de caràcters.

Noves funcions

Per a fer aquest programa caldrà definir més funcions. Primer, una que retorni un nombre complex elevat a un enter (per a aplicar els polinomis). Per fer-ho partim de la següent fórmula (*veure pàg. 25*):

$$z^n = r^n[\cos(n \cdot \theta) + i \sin(n \cdot \theta)]$$

r és la norma de z; θ es pot obtenir si sabem que $\theta = \pm\text{acos}\,(z.real/r)$ i té el mateix signe que la part imaginària. D'aquesta manera, si a la nostra funció hi entrem z i n quedaria:

```
68  complex expz(complex z, int n){
69      double theta;
70      complex exp;
71      double r, h;
72
73      r = norma(z);
74      h = pow(r, n);       // Nou radi.
75
76      theta = acos(z.real/r); // Angle (funció arccosinus)
77
78      if(z.imag < 0){
79          theta *= -1;     // Si la part imaginària és negativa,
80      }                    // theta també ho serà
81
82      exp.real = h * cos(n * theta);
83      exp.imag = h * sin(n * theta);
84
85      return exp;
86  }
```

Per no omplir el `main()`, definirem també funcions per a la derivada, el polinomi i el mètode en sí, només coneixent el grau i els coeficients del polinomi (com a variable de punters).

La funció polinomi consisteix en consisteix iterar a tots els coeficients del polinomi i operar. Com que és la funció avaluada en un punt **z**, necessitarem aquest punt. Amb el grau del polinomi anirem sumant **z** elevada a un exponent (amb la funció que acabem de definir) fins a arribar al grau de la funció. Llavors la funció retornarà la suma total.

```
100 complex pol(complex z, double *polin, int grau){
101     int i;
102     complex fun;     // Variable on guardarem la suma del polinomi.
103
104     fun.real = polin[0];
105     fun.imag = 0;    // La iniciem amb el valor del terme independent.
106
107     for(i=1; i<=grau; i++){
108         fun = suma(fun, escalar(polin[i], expz(z, i)));
109     }
110     /* Sumem el nombre elevat un exponent i multiplicat
111      pel coeficient que li correspon, fins a arribar
112      al grau del polinomi. */
113
114     return fun;      // Retornem el resultat
115 }
```

La funció per a la derivada segueix la mateixa lògica, però ara hem de multiplicar pel grau i elevar al grau menys u (són les normes de derivació).

```
117 complex der(complex z, double *polin, int grau){
118     int i;
119     complex deriv;  // Variable on guardarem la suma de la derivada.
120
121     deriv.real = polin[1];
122     deriv.imag = 0; // La iniciem amb el valor del 1r terme.
123
124     for(i = 2; i <= grau; i++){
125         deriv = suma(deriv, escalar(i * polin[i], expz(z, i-1)));
126     }
127     /* Sumem el nombre elevat un exponent, multiplicat pel
128      coeficient i pel grau que li correspon, fins a arribar
129      al grau del polinomi. */
130     return deriv;
131 }
```

Tenint aquestes dues funcions i les d'aritmètica complexa, el mètode resulta relativament senzill. Necessitem les mateixes variables que amb les altres funcions, i a més, la tolerància per saber quan deixar d'iterar.

```
121 complex newton(int grau, complex z, double *poli, double tol){
122     complex z0;
123     z = resta(z, divi(pol(z, poli, grau), der(z, poli, grau)));
124
125     do{
126         z0 = z;
127         z = resta(z, divi(pol(z, poli, grau), der(z, poli, grau)));
128     }while(norma(pol(z, poli, grau)) > tol);
129
130     return z;
131 }
```

La condició d'escapament serà si el polinomi avaluat en el punt és més proper a zero que la tolerància, i per tant és una aproximació de l'arrel prou bona, es deixi d'iterar.

També ens ajudarà una funció que comprova si un nombre **`complex`** `c` apareix dins d'una variable de punters **`complex`** `*aux`, que té **`int`** `d` nombres emmagatzemats. La farem servir per destriar les arrels no-repetides. Caldrà també, afegir la llibreria `<stdbool.h>` a la capçalera del programa.

```
171 bool check (complex *aux, complex c, int d){
172     int j;
173     double tol = 1e-9;
174     for(j=0; j<d; j++){
175         if(norma(resta(aux[j], c)) < tol){
176             return false;
177         }
178     }
179     return true;
180 }
```

Programa

Ara la primera part del programa és declarar i iniciar totes les variables que farem servir.

```
181  int main(void){
182      double PI = acos(-1.0); /* Constant pi (3.1415...) */
183      double epsilon = 1e-10;  // Tolerància
184      double r, theta;        // Radi i angle dels punts a iterar
185
186      double *poli;       /* Variable de punters on es guardaran
187                             els coeficients del polinomi */
188                          /* Variable de punters on es guardaran
189                             els punts a iterar */
190      complex *conv, *arrels;
191
192      char nom[20];       //Nom de l'arxiu on es guardaran
193                          //les solucions.
194      complex zIt;
195
196      int d, s, N, v, j;
197      int i, l = 0;    //Variables auxiliars
198
199      FILE *sol;          /* Variable "arxiu" */
```

El següent que cal fer és demanar a l'usuari el grau del polinomi i calcular el nombre de cercles i el nombre de punts en cada cercle segons el teorema de l'apartat anterior.

```
201      printf (" Quin es el grau del polinomi?\n");
202      scanf("%d", &d);    /* Demanem el grau del polinomi */
203
204      s = ceil(0.26632*log(d));   // Calculem el nombre de cercles i
205      N = ceil(8.3247*d*log(d));  // els punts per cercle, amb les
206                                  // fórmules del teorema.
207
```

Un cop sabem `N` i `s`, podem inicialitzar les variables de punters, ja que en podem conèixer la "mida" (el nombre d'índexs).

La sintaxi és: `nom_del_punter = (tipus*)malloc((mida)*sizeof(tipus));` El tipus serà de `double` per als coeficients i `complex` per a les arrels. La mida del primer és el grau del polinomi més 1 (el terme independent). La del segon és el nombre de cercles multiplicat pels punts per cercle.

```
208      /// Inicialitzem les variables del polinomi i dels punts ///
209      poli = (double*)malloc((d+1)*sizeof(double));
210      conv = (complex*)malloc((N*s)*sizeof(complex));
211      arrels = (complex*)malloc((d)*sizeof(complex));
```

Ara inicialitzem l'arxiu i l'obrim per escriure-hi.

```
213      printf("\n Nom de l'arxiu on es guardaran les solucions:\n");
214      scanf("%s", &nom);          // Es guardarà un arxiu amb aquest nom
215                                  // a la carpeta del projecte.
216      sol = fopen(nom, "w");      // Creem l'arxiu i l'obrim per escriure-hi.
```

I ja podem demanar els coeficients i anar guardant-los a la variable de punters.

```
218  printf("\n Digues els coeficients\n"
219         "(desde el terme independent fins al major exponent):");
220
221  printf("\n   -Terme independent: ");
222  scanf("%le", &poli[0]);
223
224  for (i = 1; i <= d; i++){
225      printf("   -Coeficient de x^%d: ", i);
226      scanf("%le", &poli[i]); // Demanem els coeficients a l'usuari i els
227  }                           // guardem dins la variable "poli". Poden ser
228                              // qualsevol nombre real.
```

Si no s'haguessin guardat bé o l'últim coeficient fos 0 (condició que no es pot donar, ja que si fos així, seria un polinomi de grau inferior), saltaria un missatge d'error i s'acabaria el programa. En cas contrari, demanem paciència i comencem a iterar els punts.

```
230     if(poli == NULL || poli[d] == 0){
231         printf("Error");          // Si no s'han guardat bé o l'últim coeficient
232         return 1;                 // és 0 (cosa que no pot passar) sortirà un
233     }                             // missatge d'error.
234
235     printf("\n D'acord, espera un moment...\n");
```

Ara, per a cada punt aplicarem el mètode de Newton i el guardem a una de les variables de punters.

```
237     for(v = 1; v <= s; v++){
238         r=(1+sqrt(2))*pow((d-1.)/d, ((2*v-1.)/4*s));
239         // Calculem el radi de cada cercle.
240
241         for(j = 0; j <= N-1; j++){
242             theta = 2 * PI * j/N;
243             // I els angles de cada punt.
244
245             zIt.real = r*cos(theta);     // Declarem els punts amb
246             zIt.imag = r*sin(theta);     // radi i l'angle.
247
248             conv[l] = newton(d, zIt, poli, epsilon);
249             /* Iterem els punts i els guardem */
250             l++;
251         }
252     }
```

Després he afegit unes línies que permeten escriure el polinomi inicial al nou fitxer. Si no es posa aquesta part no s'altera el funcionament ni el resultat del programa.

```
255     fprintf(sol, "Les arrels del polinomi:\n\n");
256
257     /// Escrivim en l'arxiu el polinomi ///
258     int gr=d;
259     fprintf(sol," %.4f x^%d ", poli[gr], gr);
260
261     for(gr = d-1; gr >= 2; gr--){
262         if(poli[gr] != 0){
263             if(poli[gr] > 0){
264                 fprintf(sol,"+ %.4f x^%d ", poli[gr], gr);
265             }else{
266                 fprintf(sol,"- %.4f x^%d ", fabs(poli[gr]), gr);
267             }
268         }
269     }
271     if(poli[1] != 0){
272         if(poli[1]>0){
273             fprintf(sol, "+ %.4f x ", poli[1]);
274         }else{
275             fprintf(sol, "- %.4f x ", fabs(poli[1]));
276         }
277     }
278     if(poli[0] != 0){
279         if(poli[0]>0){
280             fprintf(sol, "+ %.4f \n\n son:\n", poli[0]);
281         }else{
282             fprintf(sol, "- %.4f \n\n son:\n", fabs(poli[0]));
283         }
284     }
```

Ara haurem de destriar aquelles convergències que hi apareguin més d'una vegada i guardar-les en un altre lloc. Utilitzarem la funció que hem definit com a **`check()`**. La d'índex 0 la podem agafar, ja que segur que serà nova. Després hem d'anar al següent índex del punter i mirar si és nova. Si ho és, l'escrivim al punter definitiu **`arrels`**, si no, passem a comprovar la següent convergència.

```
290     /* Escrivim les arrels que siguin diferents
291     entre sí (segons la tolerància) */
292     int ind = 1;
293
294     arrels[0] = conv[0];
295
296     for(j=1; j < s*N; j++){
297         if(check(arrels, conv[j], ind)){
298             arrels[ind] = conv[j];
299             ind++;
300         }
301     }
```

La variable **ind** guarda el nombre d'arrels diferents que s'han trobat.

Amb aquesta variable, escrivim totes les arrels tenint en compte que, si és més gran que el grau del polinomi, hi haurà arrels repetides o males convergències (*veure pàg. 49*). Si és igual al grau, s'han trobat totes les arrels. I si és més petita, pot haver-hi arrels dobles del polinomi, o $0 + 0\,i$ és una arrel, i podria donar error. Per a cada possibilitat, s'escriurà un missatge personalitzat que ho expliqui a l'usuari.

```
303     if(ind > d){
304         fprintf(sol, "\n ATENCIO!"
305                 "\n S'han trobat mes arrels de les esperades (total de %d)."
306                 "\n Pot ser que n'hi hagi de repetides,"
307                 " o que en surtin aproximacions.", ind);
308
309         for(j = 0; j < ind; j++){
310             fprintf(sol, "\n Arrel %d:", j+1);
311             fprintz(sol, arrels[j]);
312         }
313     }
314     if(ind < d){
315         for(j=0; j < ind; j++){
316             fprintf(sol, "\n Arrel %d:", j+1);
317             fprintz(sol, arrels[j]);
318         }
319         fprintf(sol, "\n Potser hi ha arrels dobles, o 0+0i sigui una arrel");
320     }
321     if(ind == d){
322         for(j=0; j < ind; j++){
323             fprintf(sol, "\n Arrel %d:", j+1);
324             fprintz(sol, arrels[j]);
325         }
326     }
```

Ara ja podem tancar l'arxiu i acabar el programa.

```
328     fclose(sol);    // Tanquem l'arxiu.
329
330     /// Missatge de comprovació ///
331     printf("\n Proces acabat\n");
332
333     return 0;
334 }
```

La funció **fprintz()** també s'ha de declarar i definir, però només escriu un nombre complex. Ajuda a veure el codi més net, i l'he feta perquè era més pràctic:

```
148 void fprintz(FILE *sol, complex z){
149     double tol = 1e-8;
150
151     if(fabs(z.imag)>tol){
152         if(z.imag > 0){
153             fprintf(sol, " %.8f + %.8f i",z.real, z.imag);
154         }else{
155             fprintf(sol, " %.8f - %.8f i", z.real, fabs(z.imag));
156         }
157     }else{
158         fprintf(sol, " %.8f", z.real);
159     }
160     return;
161 }
```

8.2.1 Errors del programa

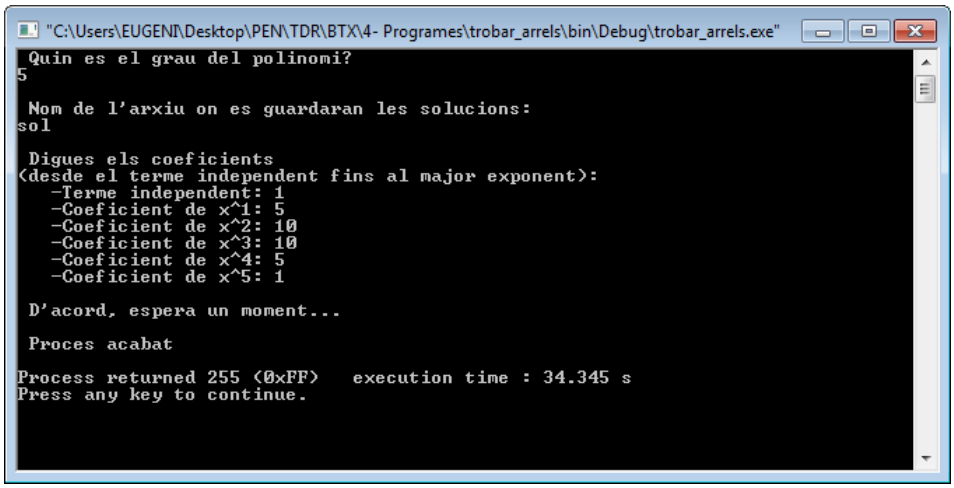
A partir d'introduir polinomis al programa, s'ha pogut detectar un petit error quan el polinomi presenta arrels múltiples. La raó pot ser que està pensat perquè per a un polinomi de grau d en trobi d arrels, però n'hi ha que no compleixen això. Per exemple, $p(z) = (z+1)^5$ només en té una.

En aquests casos, hi ha cops que les iteracions no convergeixen prou ràpid, i arriba a una aproximació de l'arrel que introduïda al polinomi dona un resultat més petit que la tolerància.

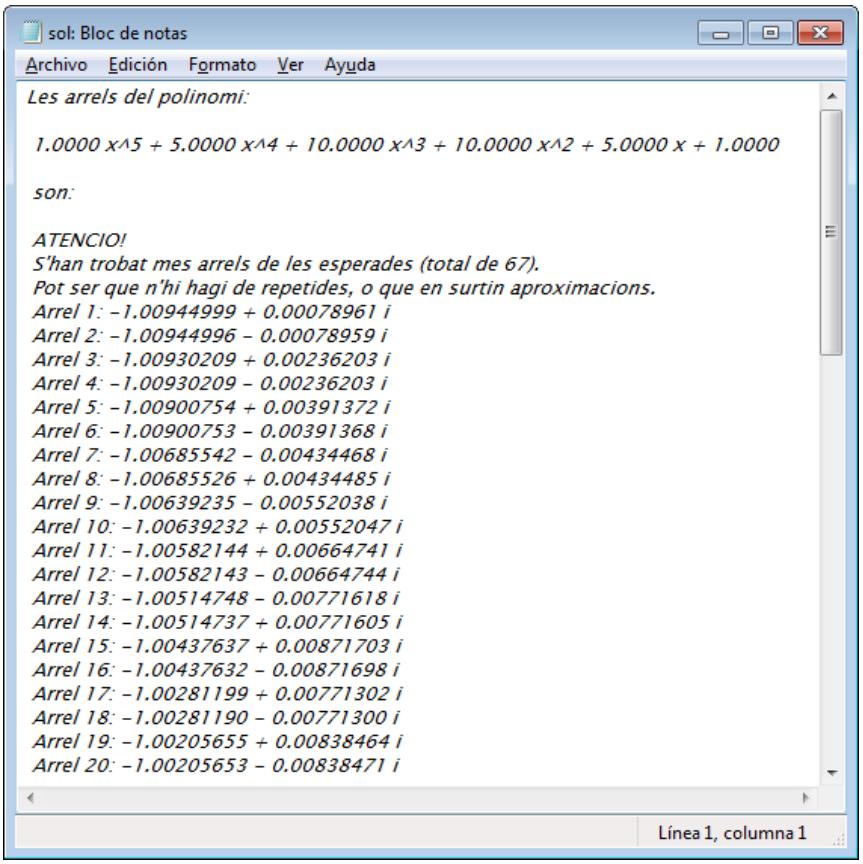
Seguint amb l'exemple anterior, $\|p(-1.0094 + 0.0007\,i)\| < 10^{-9}$. Llavors en arribar a aquest punt, es deixa d'aplicar el mètode (que és el que se suposa que ha de fer). Per tant, cal recordar que el programa retorna **aproximacions**, i és capaç de distingir-les només donada una determinada tolerància. Augmentar la tolerància seria una opció, però el temps que tardaria a executar-ho augmenta degut a la limitació en la capacitat de càlcul de l'ordinador, ja que és un ordre de magnitud més precís.

Aquest problema es fa més visible a mesura que augmentem el grau dels polinomis.

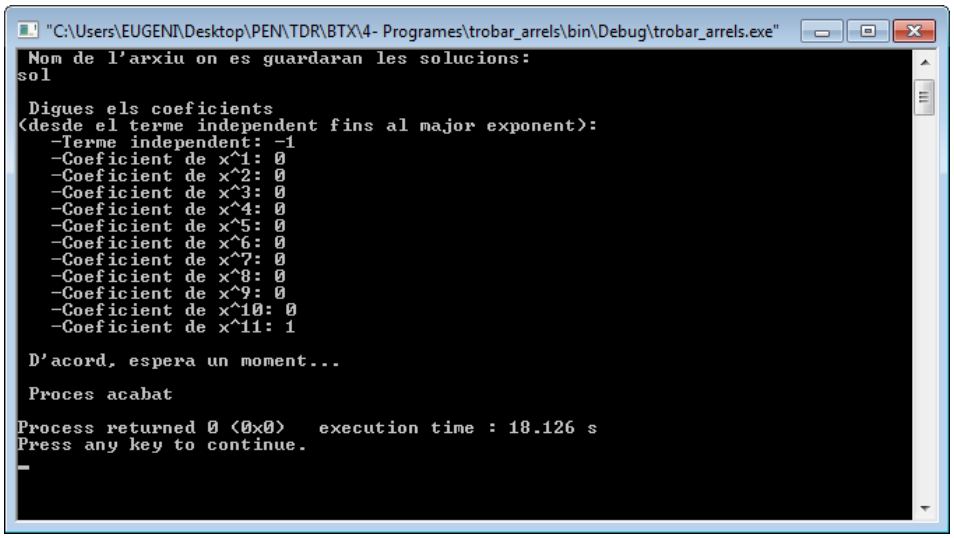
L'arxiu després d'analitzar $p(z) = (z+1)^5 = z^5 + 5z^4 + 10z^3 + 10z^2 + 5z + 1$, és el següent:

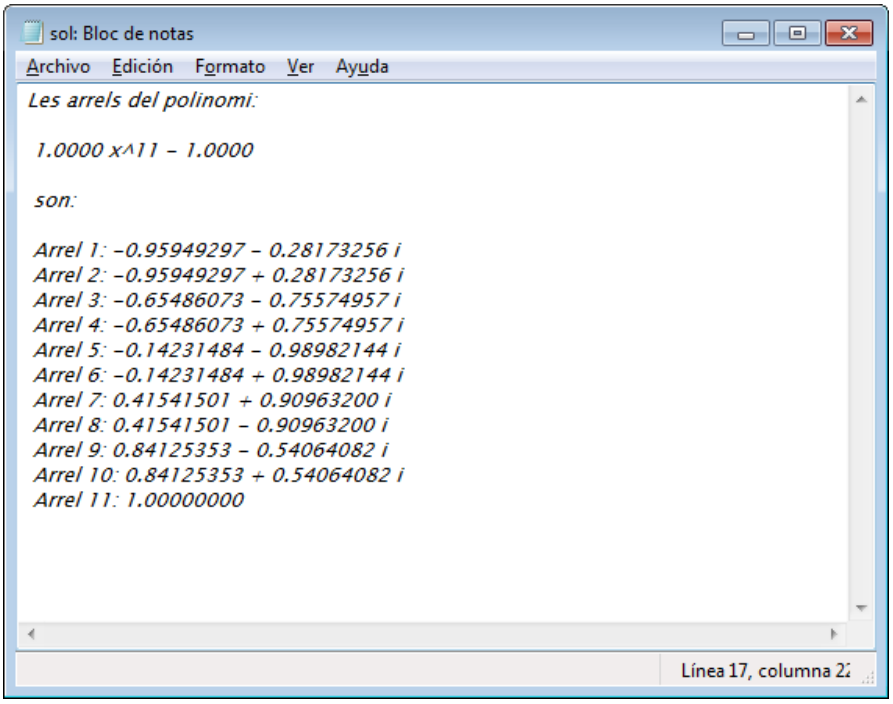


Si tornem a mirar la taula de la *pàgina 42*, es veu que per a grau 5, s'iteren 67 punts, i el programa troba 67 arrels diferents. Per tant, tots els punts arriben a convergències diferents, tot i que només hi ha una solució.



En canvi, si $p(z) = z^{11} - 1$, un polinomi amb grau més gran, però amb les arrels repartides, les troba totes (i més ràpidament).





9.- CONCLUSIONS

Els objectius del treball eren fer un treball entenedor i rigorós, i un programa.

El primer s'ha complert força bé. Al llarg de tot el treball, tot i que alguns dels conceptes són de nivell de 2n de batxillerat cap amunt, s'ha intentat explicar de la manera més entenedora possible, sempre que s'ha pogut s'han inclòs imatges (de font pròpia) per poder visualitzar el que s'explicava.

Això fa que es pugui seguir i permet adquirir els coneixements necessaris per entendre i gaudir del resultat final.

La part dels programes s'ha assolit també satisfactòriament, però no del tot. És cert que s'ha fet un programa que dibuixa les fractals i que és molt versàtil en quant a entrar-hi diferents funcions (*veure annex*), però a l'hora de voler generalitzar-ho per a qualsevol polinomi, hi havia vegades que no sortia bé (*veure pàg. 48*).

L'inconvenient que té és que cal saber les arrels del polinomi abans d'entrar-lo, i que s'ha d'entrar a mà. El programa que hauria de dir les arrels del polinomi no té una fiabilitat del 100%, per tant no es pot implementar i fer una combinació dels dos grans programes del treball. D'aquesta manera només introduint el polinomi retornaria el pintat del pla. No és fàcil, per això no em preocupa no haver-ho aconseguit.

De fet, si s'introdueix tot a mà, els resultats són els esperats, per tant es pot dir que és funcional, però requereix feina prèvia de l'usuari.

Programar és un procés que de vegades dona errors que costen d'explicar. El fet de no estar gaire familiaritzat amb el llenguatge fa que es puguin cometre errades sense adonar-se'n (en temes de sintaxi sobretot, però també en l'estructura dels programes). Però la majoria de programes funcionaven i és tan sols el projecte més ambiciós el que falla.

Un repte per seguir amb la recerca seria aconseguir aquest programa amb el qual s'introdueix un polinomi qualsevol, en calcula les arrels i retorna el pintat del pla.

Ara bé, estic molt content de com ha sortit, i de tot el que he après també. Tenint en compte que quan el vaig començar no havia fet encara mai una derivada, el que s'ha acabat tractant és bastant més complicat.

Era un treball que tocava molts camps, i això ha fet que m'hi interessés de seguida, però també que el que es tracta sigui complicat, i les dificultats que això comporta. Però crec que és més aviat un argument a favor, ja que el fet que sigui una part de les matemàtiques desconeguda per mi suposa un repte: et descobreix un món nou.

És el que el fa atractiu. Que hi hagi moltes fórmules o que sigui complicat no hauria de fer enrere ningú.

10.- FONTS D'INFORMACIÓ

FAGELLA RABIONET, Núria & JARQUE I RIBERA, Xavier. Enseñar y saber en el siglo XXI (2007). *Iteración compleja y fractales: matemáticas y estadística*. Institut de Ciències de l'Educació. Barcelona.

HUBBARD, John; SCHLEICHER, Dierk; SUTHERLAND, Scott. (1998). *How to really find roots of polynomials by Newton's method.* Inventiones mathematicae, Springer, Berlín.

ROSSROESSLER. Roessler, Ross. 2005. *"History of Complex Numbers (also known as History of Imaginary Numbers or the History of i)"* [en línia] <https://rossroessler.tripod.com/> [Consulta: 20 maig 2020]

SK33LZ. Moore, Jason. *"History of fractals"* [en línia] <https://sk33lz.com/create/fractals/history-fractals> [Consulta: 30 juny 2020]

VIQUIPÈDIA. *"Gaston Julia"* [en línia] <https://ca.wikipedia.org/wiki/Gaston_Julia> [Consulta: 2 juliol 2020]

WIKIPEDIA. *"Benoît Mandelbrot*" [en línia] <https://en.wikipedia.org/wiki/Benoit_Mandelbrot> [Consulta: 2 juliol 2020]

WIKIPEDIA. *"Calculus"* [en línia] <https://en.wikipedia.org/wiki/Calculus> [Consulta: 27 juny 2020]

WIKIPEDIA. "*Method of Fluxions*" [en línia] <https://en.wikipedia.org/wiki/Method_of_Fluxions> [Consulta: 16 maig 2020]

WIKIPEDIA. *"Newton's method"* [en línia] <https://en.wikipedia.org/wiki/Newton%27s_method> [Consulta: 16 maig 2020]

WIKIPEDIA. *"Pierre Fatou"* [en línia] <https://en.wikipedia.org/wiki/Pierre_Fatou> [Consulta: 2 juliol 2020]

WIKIPEDIA. *"Gran peste de Londres"* [en línia] <https://es.wikipedia.org/wiki/Gran_peste_de_Londres> [Consulta: 15 maig 2020]

WIKIPEDIA, "Fractal" [en línia] < https://es.wikipedia.org/wiki/Fractal > [Consulta: 30 juny 2020]

WOLFRAM ALPHA, *"Computational intelligence"* [en línia] < https://www.wolframalpha.com/ > [Consulta: 24 agost 2020]

XATAKA CIENCIA. Parra, Sergio. "*Vivir en cuarentena fue un estímulo para que Newton cambiara la historia de la Física"* [en línia] < https://www.xatakaciencia.com/sabias-que/vivir-cuarentena-fue-estimulo-newton-cambiara-historia-fisica > [Consulta: 15 maig 2020]

També he comptat amb l'ajut d'un alumne de la Facultat de Matemàtiques de la UB, David Carrasco (i del suport de la Dra. Fagella). Dels apunts dels seus professors ha sortit gran part de la informació coberta en el treball:

- Dr. TATJER, Juan Carlos. *Mètodes Numèrics I*. Universitat de Barcelona. Barcelona. 2018.
- Dr. NAVARRO, Vicente. *Topologia*. Universitat de Barcelona. Barcelona. 2019.
- Dra. FAGELLA, Núria. *Models Matemàtics i Sistemes Dinàmics*. Universitat de Barcelona. Barcelona. 2019.
- Dr. GARCÍA, Ricardo. *Topologia Global de Corbes i Superfícies*. Universitat de Barcelona. Barcelona. 2020.