Nova Scientia". *Teaching innovations 27* (3) (2014), 114–126.

[10] M. R. Massa-Esteve; A. Roca-Rosell; C. Puig-Pla. "Mixed Mathematics in engineering education in Spain: Pedro Lucuce's course at the Barcelona Royal Military Academy of Mathematics in the eighteenth century". *Engineering studies* (3) (2011), 233–253.

[11] A. Massoti. "Niccolò Tartaglia". C. C. Gillispie (ed.). *Dictionary of scientific biography.* Nova York: Scribner's 13 (1971–1991), 258–262.

[12] L. Roberts; S. Schaffer; P. Dear (ed.). *The mindful hand: Inquiry and invention from the late Renaissance to early industrialisation.* Amsterdam: Koninklijke Nederlandse Akademie van Wetenschappen (2007).

[13] S. Rommevaux; M. Spiesser; M. R. Massa-Esteve (dir.). *Pluralité de l'algèbre à la Renaissance.* París: Honoré Champions Editeur (2012).

[14] P. L. Rose. *The Italian Renaissance of Mathematics: Studies on Humanists and Mathematicians from Petrarch to Galileo.* Ginebra: Libraire Droz (1975).

[15] N. Tartaglia. *Euclide Megarense Philosopho, solo introduttore delle Scientie Mathematice. Diligentemente rassettato, et alla integrità ridotto, per il degno professore di tal Scientie Nicolo Tartalea Brisciano secondo le due tradottioni.* Venetia: Apresso Curtio Troiano (1565).

[16] N. Tartaglia. *Nova Scientia*, (1537).

[17] N. Tartaglia. *La Nueva Ciencia (R. Martínez i C. Guevara, trad.).* Col·lecció MATHEMA. Ciutat de Mèxic: Facultat de Ciències, UNAM (1998).

[18] M. Valleriani. *Metallurgy, Ballistics and epistemic instruments. The Nova Scientia of Niccolò Tartaglia. A new edition.* Max Planck Research Library for the History and Development of Knowledge. Sources 6. Berlín: Edition Open Access (2013).

Bits de matemàtiques

Mani'm?

Laura Brustenga i Moncusí, UCPH
Martí Prats i Soler, UB

El "Bits" de la *SCM/Notícies* 49, el vam dedicar a fer un repàs de continguts divulgatius en línia.

El "Bits" d'aquest número té com a objectiu exposar una eina útil per a la divulgació. Es tracta d'un programa de creació de vídeos matemàtics anomenat Manim. Quin nom més apropiat per a un llenguatge de programació! És com si ens demanés a crits que comencem a fer divulgació matemàtica en línia en català, no us sembla?

Com sempre, animem els lectors a fer-nos arribar propostes per poder-les disseccionar en aquest apartat. Envieu-les a: brust@mat.uab.cat.

Manim (acrònim de Mathematical Animation Engine) és un motor Python per programar animacions precises, especialment dissenyat per a vídeos de matemàtiques. Va ser creat per Grant Sanderson per a les animacions del seu canal de divulgació @3blue1brown. Actualment n'hi ha dues versions: ManimGL[8], mantinguda pel creador original, i la versió comunitària, Manim[9]. En aquest article parlarem de la versió comunitària, recomanada per a la creació de continguts. Es pot provar el programa en línia a través d'uns fulls Jupyter[10]. De tota manera, la instal·lació és prou senzilla si estem familiaritzats amb el llenguatge Python; només cal seguir les instruccions de la pàgina de Manim.

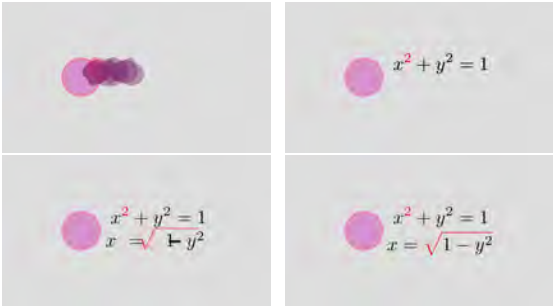
[8] https://github.com/3b1b/manim
[9] https://github.com/ManimCommunity/manim/
[10] https://try.manim.community/

Exemples

Tot seguit expliquem alguns exemples per il·lustrar la potència del programa en qüestió. En primer lloc, el codi següent

```
class AnimatedCircleToTex(Scene):
    def construct(self): #Constructor
        # Definicio dels objectes
        circle = Circle()
        text = MathTex(r"x^{{2}}+{{y^2}}={{1}}",
       font_size=96)
        text2 = MathTex(r"x{{=}}{{\sqrt}}{1{{-}}y^2}",
       font_size=96)
        # Una mica de color
        circle.set_fill(PINK, opacity=0.5)
        text.submobjects[1].set_color(YELLOW)
        text2.set_color_by_tex("\sqrt", YELLOW)
        # Emplacement dels objectes
        text2.next_to(text, DOWN)
        VGroup(text,text2).next_to(circle, RIGHT)
        VGroup(circle,text,text2).move_to(ORIGIN)
        # Creacio animacio
        self.play(Create(circle))
        self.play(ReplacementTransform(circle.copy(), text))
        self.play(TransformMatchingTex(text.copy(), text2,
       transform_mismatches=True,key_map={"1":"1","+"
       ":"−", "2": "\sqr"}))
        self.wait()
```

genera una animació on apareix un cercle que es converteix en l'equació de la circumferència, amb l'exponent de la primera variable de color vermell, i després aïlla la x.



Instantànies de l'animació de transformació d'un cercle en equació.

L'ús de dobles claudàtors dins del codi LaTeX genera *substrings* separades que després es poden pintar d'un color diferent; i es poden relacionar les unes amb les altres per fer la transició animada.

Notem que la classe `AnimatedCircleToTex` conté el codi de l'animació dins de `construct`. Aquest codi es desa dins d'un arxiu de text que identifiquem amb l'extensió de Python, com ara "scene.py". Per renderitzar l'escena, cal entrar al terminal, navegar fins a la carpeta que conté l'arxiu creat i executar

```
manim −pql scene.py AnimatedCircleToTex
```

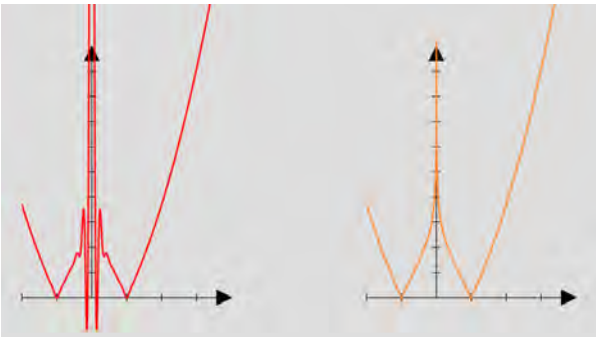
El marcador "p" indica que visualitzarem el resultat quan s'hagi creat l'animació i el "ql" indica qualitat baixa. Si volem més qualitat, podem posar "qm", "qh" o "qk". L'arxiu pot contenir tantes escenes com es vulgui; a l'hora d'executar el programa cal escollir quina es renderitza.

Manim genera animacions amb el fons negre per defecte. Per canviar aquest color de fons, hem creat un arxiu de text pla anomenat "manim.cfg" a la mateixa carpeta que conté l'animació amb el codi següent

```
background_color = #DDDDDD
```

Un dels grans avantatges de Manim és que es poden afegir fàcilment diversos sistemes de coordenades a la pantalla, i treballar amb cada un per separat.

```
class DosEixos(Scene):
    def construct(self):
        eix1 = Axes(x_range=[−2,4], y_range=[0,10],
       x_length=5).set_color(BLACK)
        eix2 = eix1.copy()
        eix1.to_edge(LEFT)
        eix2.to_edge(RIGHT)
        eixos= VGroup(eix1,eix2)
        #Funcio a dibuixar
        def func(x):
            return abs(np.log(abs(x))+x**2−1)
        corba1 = eix1.plot(func, color=PURE_RED)
        corba2 = eix2.plot(func, x_range=(−1.9999, 4,
       0.002), color=ORANGE)
        corbes = VGroup(corba1,corba2)
        self.add(eixos,corbes)
```

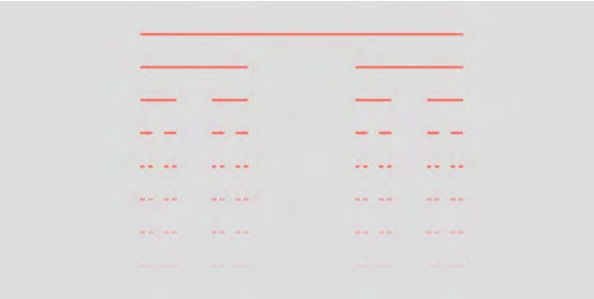


Imatge generada amb la classe `DosEixos`.

Notem que per crear les gràfiques Manim treballa amb corbes de Bézier, i a vegades genera resultats no desitjats. Per afinar la punteria convé especificar el rang com hem fet a la segona gràfica. La tercera coordenada indica el pas, i l'hem escollit garantint molts punts sense passar per la singularitat.

Presentem ara una animació amb més gràcia: la creació del conjunt de Cantor fins a ordre 8.

```
class CantorSet(Scene):
    def construct(self):
        #Inicialitzem eixos de coordenades i variables
        ax = Axes(x_range=[−0.5,1.5], y_range=[−0.2, 1.2])
        m=7; lpre=[]; lpost=[]
        for j in range(m+1): lpre.append([]); lpost.append([])
        #Iteracio del conjunt de Cantor
        def cantorize(p1,p2,o):
            lpre[o].append(Line(ax.c2p(p1[0],p1[1]+1./m),ax.c2p(p2[0],p2[1]+1./m),color=GREEN))
            lpost[o].append(Line(ax.c2p(p1[0],p1[1]),ax.c2p(p2[0],p2[1]),color=RED))
            if o<m: #Iteracions en forma d'arbre:
                cantorize(p1 + (0,−1./m),p1+(p2−p1)/3.+(0,−1./m),o+1)
                cantorize(p1+(p2−p1)*2/3.+(0,−1./m),p2+(0,−1./m),o+1)
        #Creacio del conjunt de Cantor
        cantorize(np.array([0,1]),np.array([1,1]),0)
        #Creacio animacio:
        #apareixen successivament les linies grogues (lpre)
        #i es converteixen en les taronges (lpost)
        #creant efecte de ''pluja de Cantor''
        for k in range(len(lpost)):
            mpre=VGroup(*lpre[k])
            mpost=VGroup(*lpost[k])
            self.play(FadeIn(mpre))
            self.play(ReplacementTransform(mpre,mpost))
            self.wait()
```

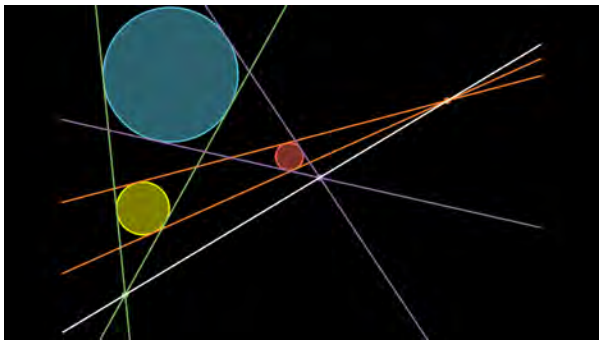


Darrer fotograma de l'animació `CantorSet`.

Sobre aquest darrer codi, convé comentar els elements. Creem uns eixos *ax*, *m* indica el nombre de passos de la iteració, i *lpre*, *lpost* seran vectors de vectors de línies. Cada un dels vectors *lpre[j]* conté les línies verdes que constitueixen la iteració j-èsima del conjunt de Cantor, situada encara sobre la iteració anterior. El vector *lpost[j]* és de color vermell i conté els mateixos segments, però ja situats en la seva posició final. A l'hora d'executar `ReplacementTransform`, creem un moviment interpolat entre l'un i l'altre. La instrucció `ax.c2p` agafa les coordenades que li donem i retorna la posició en pantalla de l'element que volem col·locar als eixos ax. La resta del codi s'explica força sol.

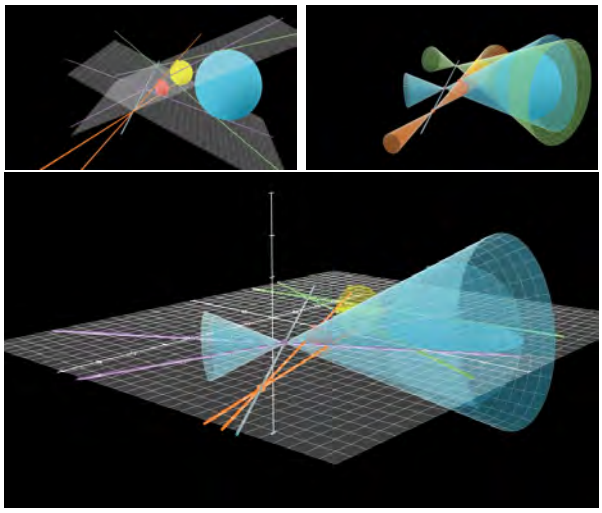
Creant un vídeo divulgatiu

El propòsit real de Manim és la creació de vídeos divulgatius. Per posar-lo a prova, hem creat un vídeo sobre geometria, concretament per explicar el teorema de Monge i la seva demostració. En el pla, tres cercles qualssevol determinen tres parells de tangents exteriors comunes a cada parell de cercles. El teorema diu que les interseccions de cada un d'aquests parells cauen sobre una mateixa recta.



Teorema de Monge.

El més interessant del teorema és la demostració utilitzant les tres dimensions. Es tracta d'entendre les circumferències com els equadors de tres esferes. Aleshores les tangents comunes esdevenen rectes oposades dels cons tangents als parells d'esferes, i cada un dels tres cons conté en particular una recta de cada pla tangent exterior a les tres esferes. D'aquí se'n dedueix que els vèrtexs dels tres cons estan a la intersecció dels dos plans, que és una recta, i el teorema segueix immediatament.



Instantànies de la demostració del teorema de Monge.

El codi és massa llarg i potser el seu coeficient d'interès per línia és massa baix per incloure'l en aquesta revista, així que el lector assedegat el pot trobar al nostre repositori en línia[11], on també trobarà els altres codis explicats en aquest article i les corresponents animacions obtingudes.

Sí que ens interessa compartir les lliçons obtingudes amb el procés, amb el benentès que el nostre coneixement de Manim és superficial i, per tant, les conclusions poden ser conseqüència de la nostra poca traça.

En primer lloc, un cop creat el vídeo, cal tenir algun programa que permeti incorporar-hi so, i cal mesurar bé la durada de cada element de l'animació perquè després tot casi. Hom pot gravar primer el so i després fer l'animació a mida, o just a la inversa. L'addició del so al video es pot fer per línia de comandes usant `ffmpeg`, per exemple, o bé amb algun programa d'edició de vídeo com `iMovie`, `Movie Maker` o `Openshot`.

En segon lloc, si bé l'elegància de les animacions fa les produccions molt atractives, el llenguatge és molt menys complet, ara com ara, que el GeoGebra a l'hora de manipular objectes geomètrics. Cal picar codi del dur si es volen aconseguir resultats on els objectes generats es moguin de manera dinàmica tal com ho fan al GeoGebra; la classe Circle, per exemple, no disposa ni tan sols d'un constructor basat en centre i radi, ni d'accés a les seves equacions. Obtenir interseccions i tangents comunes suposaria haver de crear tota l'estructura de classes de cap i de nou, cosa que a GeoGebra ja tenim resolta. Per tot plegat, hem optat per generar tots els elements mitjançant el GeoGebra i, un cop construït tot, ho hem introduït a Manim amb les coordenades dels punts clau. La construcció final, per tant, no és dinàmica, ja que aconseguir un efecte de punt lliscant hauria suposat una inversió de temps força més gran.

A més a més, si bé el procés de renderitzatge en dues dimensions és molt ràpid, en tres dimensions és molt feixuc. Convé planificar bé i mirar de renderitzar primer cada animació per separat.

La contrapart és que tenim tot el poder de Python i de NumPy en particular. Així, disposem d'una gran capacitat per controlar el que volem crear, així com per generar construccions força complexes. Hi ha algunes creacions que serien impensables amb GeoGebra i, sobretot, la manipulació algebraica queda molt clara si es fan els vídeos a consciència, tot i que en aquest vídeo en particular no n'hem fet ús.

[11] `https://github.com/BitsDeNoticiesSCM/manim`

Geogebra

Les funcions de dues variables amb GeoGebra

Bernat Ancochea Millet
President Associació Catalana de GeoGebra

En números anteriors us vam explicar com podem construir superfícies de revolució, superfícies reglades i també superfícies d'altres tipus amb el GeoGebra. En aquest escrit us mostrem unes eines del programa molt potents per treballar amb funcions de dues variables. En aquest enllaç hi trobareu explicacions pas a pas i molts exemples, que poden ser útils per a assignatures de primer curs de graus universitaris.

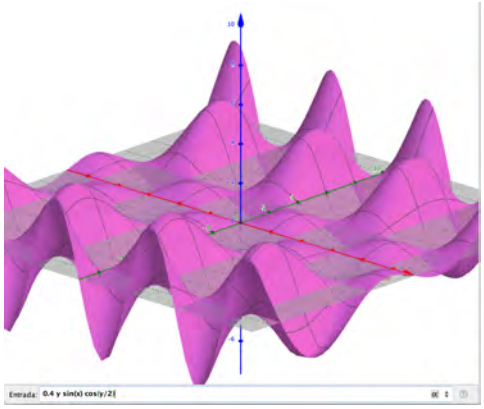


Figura 1. $f(x, y) = \ldots$