Capítulo 8

Números Complejos

El uso de números complejos puede no ser familiar para muchos de Uds. en Geociencias, pero en el tratamiento de datos y series de tiempo, se usa para el análisis de Fourier, e geofísica también es muy utilizado.

En algunos lenguajes, el uso de números complejos no es sencillo. En Fortran por ejemplo, un número complejo es estandar, mientras que en C/C++ no lo es y se requiere cargar librerías para poder trabajar con ellos. En Python se puede definir un número complejo usando

```
z = complex(2,3)
```

o también

```
z = 2+3j
```

donde `j` se usa en vez de `i` en Matlab.

Un programa básico con algunos ejemplos básicos del uso de números complejos en Python se muestra:

```
# basic_complex.py
# Some basic complex number concepts

import cmath

z0 = 1j
z1 = 1j * 1j

print(z0)
print(z1)

z0 =  complex(2,3)
z  = 2+3j
print(z, z0)
```

```
print(z.real)
print(z.imag)
print(z.conjugate())

z = complex(3,4)
print(z, abs(z))
print(pow(z, 2))

# Some function on complex
print("Functions on complex numbers")
z = complex(2,3)
zsin = cmath.sin(z)
zcos = cmath.cos(z)

print('For z = ', z)
print('cos(z), sin(z)', zcos, zsin)
```

8.1. Algunos ejemplos de números complejos

Acá otro programa con interacción del usuario para multiplicar dos números complejos

```
# testcomplex.py

import cmath

zreal,zimag = input("Enter first  complex number ").split()
a = complex(float(zreal),float(zimag))

zreal,zimag = input("Enter second complex number ").split()
b = complex(float(zreal),float(zimag))

c = a*b

print('Product         = ', c)
print('abs of product  = ', abs(c))
print('sqrt of product = ', cmath.sqrt(c))
```

con resultados para la salida del programa

```
gprieto > python testcomplex.py
Enter first  complex number 1 1
Enter second complex number -0.5 0.1
```

```
Product         =  (-0.6-0.4j)
abs of product  =  0.7211102550927978
sqrt of product =  (0.24607951468254918-0.812745426038436j)
```

Los números complejos en Python se representan como dos pares de números en parentesis, el segundo con una `j`. Por esto, no es fácil pedir el `input` del usuario. Se solicita dos números y el programa los lee y los pone dentro de un número complejo. Cualquiera de los siguientes casos

```
1, 1
(1, 1j)
1+1j
```

producirían errores en nuestro programa

Python proporciona las funciones para unir dos n??meros dentro de un cmplejo, o extraer la parte real e imaginarias de un número complejo.

```
# testcomplex2.py

import cmath

zreal = input("Enter real part ")
zimag = input("Enter imaginary part ")

a = complex(float(zreal),float(zimag))
print ('z = ', a)

a = cmath.exp(a)
print ('exp(a) = ', a)

print ('Real and Imaginary parts ', a.real, a.imag)
```

con resultados

```
gprieto > python testcomplex2.py
Enter real part 1
Enter imaginary part 2
z =  (1+2j)
exp(a) =  (-1.1312043+2.4717266j)
Real and Imaginary parts  -1.1312043 2.4717266
```

8.2. Arreglos de números complejos

`NumPy` tiene la capacidad de trabajar con arreglos de números complejos, lo cual puede ser de gran utilidad. Incluso las operaciones en `numpy` trabajan sin problema (o por lo menos la parte que he mirado) con arreglos complejos.

Abajo un ejemplo simple de creeación de un arreglo complejo, y el uso de multiplicación matricial `matmul()` con arreglos complejos.

```
# array_complex.py

import cmath
import numpy as np

a = np.array([1+2j, 3+4j, 5+6j])

print('Complex array ', a.shape)
print(a)

print ('Imag part ', a.imag)
a.imag = np.array([8, 10, 12])

print('New array ', a)

a = np.array([1+2j, 3+4j, 5+6j])
a = a[:, np.newaxis]
b = a.T

print('Array a, (3,1) ', a)
print('Array b, (1,3) ', b)

c = np.matmul(a,b)

print ('matmul(a*a.T)')
print(c)

c = np.matmul(a,np.conjugate(b))

print ('matmul(a*conj(a.T))')
print(c)
```

8.3. Fractales

Si no lo han visto, en internet se encuentran figuras muy llamativas de fractales. Uno de los sets mas famosos son el *Mandelbrot Set*, el cual es relativamente fácil de generar en un programa de computador. Para hacerlo, se requiere poder manejar números complejos en Python.

La idea básica detrás del cálculo de fractales es que hay ciertas funciones complejas que, cuando se calculan repetidamente, pueden divergir o estar limitadas. El que divergan o no, es muy sensible a pequeños cambios en el valor

del número complejo que inicia el cálculo. Esto se observa muy cerca de ciertas regiones en el plano complejo.

Una de las imagenes más famosas de fratales es el *Mandelbrot Set*. Para generar este *set*, empezamos considerando un númerp complejo `c`, al cual se le aplica el siguiente algoritmo:

```
set z=0 to start
then repeateadly compute z = z*z + c
until |z| > 2 OR the number of iterations exceeds some number
output the number of iterations
```

Por ejemplo, si $c = 0{,}3 + 0{,}3i$, entonces:

```
1ra iter: z =  0.30 + 0.30i  |z| = 0.42
2da iter: z =  0.30 + 0.48i  |z| = 0.57
3ra iter: z =  0.16 + 0.59i  |z| = 0.61
4ta iter: z = -0.02 + 0.49i  |z| = 0.49
```

En este caso, `z` permanecerá limitado aún hasta después de miles de iteraciones. Sin embargo, para $c = 0{,}5 + 1{,}0i$:

```
1ra iter: z =  0.50 + 1.00i  |z| = 1.10
2da iter: z = -0.25 + 2.00i  |z| = 2.00
3ra iter: z = -3.44 + 0.00i  |z| = 3.40
4ta iter: z = 12.32 + 1.00i  |z| = 12.4
5ta iter: z = 151.1 +25.63i  |z| = 153.4
```

y el valor de `z` rápidamente explota a valores infinitos. En la iteración 10 el valor seguramente ya excederá la capacidad para que un computador pueda guardar el número en memoria. Sin embargo, podemos evitar hacer estos cálculos de `z` una vez su valor absoluto exceda 2.0, ya que se puede demostrar que una vez ese valor es alcanzado, `z` tiende a diverger.

El cálculo se realiza para una serie de valores de `c` y el resultado se grafica como función de la posición de `c` en el plano complejo (la parte real en el eje `x` la parte imaginaria en el eje `y` y el número de iteraciones como valor de amplitud). El *Mandelbrot set*, es el set de números complejos `c` para los cuales el tamaño de $z^2 + c$ es finito aun después de un número infinito de iteraciones. Una buena aproximación, es por ejemplo realizar esta operación hasta un número grande de iteraciones (1000 puede ser un buen ejemplo), y asumir que si $|z| > 2$, el valor va a diverger.

Un código para calcular el *Mandelbrot set* se muestra a continuación, donde se define de manera automática el número de puntos para el eje `x` y el eje `y` (eje de los reales e imaginarios).

```
# mandel1.py
# Plot the mandelbrot set for a number of points
#
```

```
import cmath
import numpy as np
import matplotlib.pyplot as plt

x1,x2,y1,y2 = input("Enter x1,x2,y1,y2 ").split()
x1 = float(x1)
x2 = float(x2)
y1 = float(y1)
y2 = float(y2)

# Define grid size and start matrix
nx = 100
ny = 100
dx = (x2-x1)/float(nx)
dy = (y2-y1)/float(ny)
dat = np.zeros((nx,ny))

# Main loop for each value
for ix in range(nx):
   for iy in range(ny):
      cr = x1 + dx/2. + dx*float(ix)
      ci = y1 + dy/2. + dy*float(iy)
      # Create complex number
      c = complex(cr,ci)
      z = complex(0.0, 0.0)
      for it in range(1000):
         z = c + z*z
         if (abs(z) > 2):
            break
      dat[ix,iy] = it+1

# Rotate matrix and plot log10 scale
dat = np.transpose(dat)
zdat = np.log10(dat)

# Plot result
plt.imshow(zdat,interpolation='bilinear',extent=(x1,x2,y2,y1),cmap='Spectral')
plt.axis('equal')
plt.savefig('test1.png')
plt.show()
```

La figura resultante es una figura que al hacer zoom, las características de la estructura son similares.

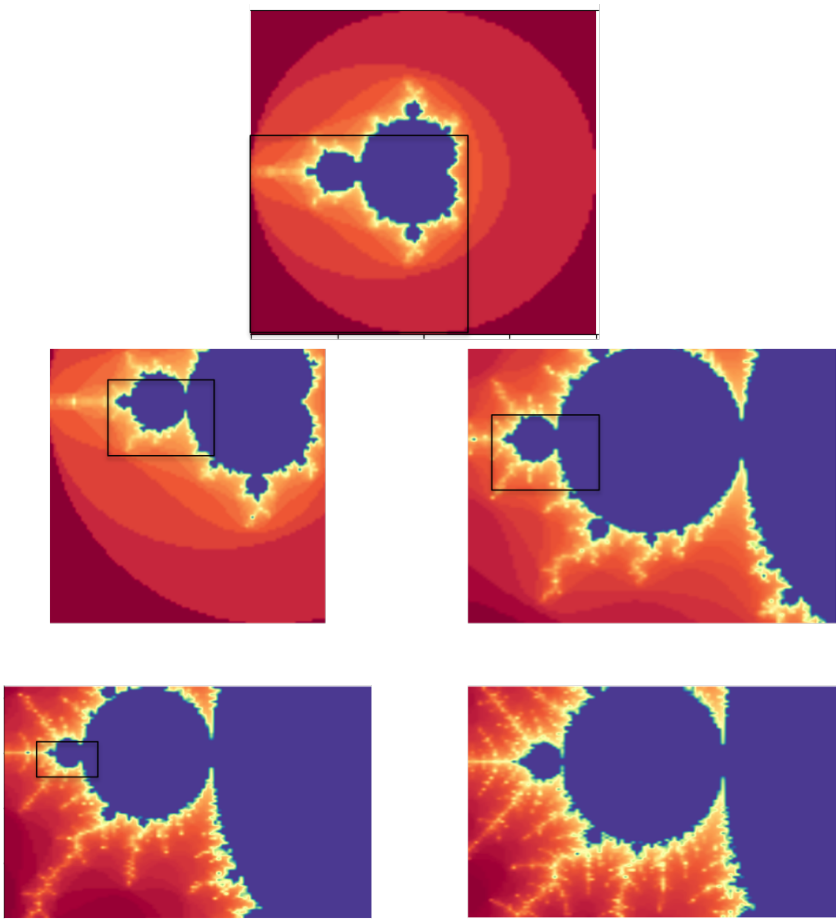

Figura 8.1: Mandelbrot set en el espacio $-2{,}0 \rightarrow 2{,}0$ (arriba) y varias aproximaciones como se muestra en el cuadro negro; resultado de correr `mandel.py` con varios valores para los límites.

Problemas

8.1. Escriba un programa para evaluar la precisión en la definición de un número complejo en Python, usando la formula

$$e^z = e^{(x+yj)} = e^x * e^{yj} = (e^x cos(y)) + (e^x sin(y))j$$

para lo cual se debe *programar* cada función para un número complejo `z` que indique el usuario. Asegurese que el programa de como resultados la respuesta correcta para

```
exp( 1.1 + 2.3j) = -2.002 + 2.240j
exp( 0.0 + 1.2j) =  0.362 + 0.932j
exp(-0.5 + 0.0j) =  0.607 + 0.000j
```

Se encuentran diferencias entre las funcion predifinida `cmath.exp()` y los resultados de las otras fórmulas?

8.2. Utilizando el programa desarrollado en clase para el *Mandelbrot set*, genere 3 figuras haciendo *zoom* en diferentes regiones. Busque figuras que sean bonitas e interesantes, no cualquier lugar. Note que el *Mandelbrot set* es auto-similar.

8.3. El artículo de *Scientific American* del autor A.K. Dewdney muestra de una manera muy didáctica como se pueden generar los fractales, y en particular el artículo de Noviembre de 1987, discute los *Julia sets* y su relación con el *Mandelbrot set*.

En resumen, para el Mandelbrot set que se muestra en el código arriba, $z = 0{,}0 + 0{,}0j$ y `c` define el punto en el plano complejo (varía según el grid que se quiere). Que pasaría si en cambio, se define el valor de `c` como valor fijo, y `z` juega el papel de punto inicial, lo que lleva al *Julia set*. Una figura distinta se espera dependiendo del valor de `c` que se elija.

Genere 3 figuras que Ud. crea que son bonitas de *Julia sets*. Marque muy claramente los valores de `c` que utilizó y los límites de su figura.