PRACTICA 1. ANÁLISIS COMPLEJO I y II.

1 Análisis Complejo I: Operaciones básicas con números complejos.

Veremos incialmente algunos comandos específicos para trabajar con números complejos. Recordemos que en Maxima la unidad imaginaria se representa por %i.

Podemos comenzar definiendo el número complejo z=3-2i:

```
--> z:3-2*%i;
```

Si necesitamos escribir la parte real de z usaremos

```
--> realpart(z);
```

mientras que para la parte imaginaria

```
--> imagpart(z);
```

Sea entonces w=3+2i . Podemos operar con ambos complejos, y así hacer:

```
--> w:-1+2*%i;
```

Notemos que a veces Maxima nos devuelve la parte imaginaria del complejo antes que la parte real. Para que siempre escriba primero la parte real seguida de la parte imaginaria, haremos:

```
--> powerdisp:true$
    w;
```

```
--> /*podemos calcular la suma o diferencia de ambos,
    que los introducimos como unas nuevas variables,
    que llamamos suma y resta*/

    suma:z+w;
    resta:z-w;
```

Podemos multiplicar complejos como se multiplican cualquier par de nros:

```
--> z*w;
```

Observamos que nos da una expresión literal, por lo que si queremos calcular su resultado haremos:

```
--> expand(%);
```

Lo mismo ocurre si intentamos dividir:

```
--> z/w;
```

y si lo que queremos es que nos de en forma cartesiana, haremos:

```
--> rectform(z/w);
```

Podemos calcular el módulo de z, usando abs(z) o cabs(z);:

```
--> abs(z);
```

```
--> cabs(z);
```

NOTA:Si usamos abs(z) para calcular el módulo del complejo z, obtendremos el valor absoluto de z, y no el resultado dado por cabs(z), que nos da (x^2+y^2)^(1/2). La opción abs(z) suele usarse cuando se utilizan cálculos simbólicos (por ejemplo, calcular límites, derivadas, integrales,...)

Para hallar el conjugado de z haremos:

```
--> conjugate(z);
```

Podemos expresar el complejo z en forma polar |z|·e^(i·arg(z)):

```
--> polarform(z);
```

Observamos que esta última forma nos da el módulo y el argumento de z. No obstante, podemos calcular directamente su argumento (anteriormente ya hemos calculado su módulo) sin necesidad de expresarlo en forma polar. Para ello usaremos:

```
--> carg(z);
```

Podemos calcular potencias naturales de z:

```
--> z^2;
    expand(%);
```

También podemos calcular potencias enteras de z:

```
--> z^(-2);
    expand(%);
    rectform(%);
```

A la hora de calcular raíces complejas, hemos de tener en cuenta que Maxima no las va a obtener todas (sabemos que todo complejo z tiene n raíces n-simas), sino que nos va a dar la primera de ellas (recordemos que las demás siempre tienen el mismo módulo pero se diferencian en el argumento):

```
--> z^(1/2);
    rectform(%);
```

```
--> z^(1/3);
    rectform(%);
```

Podemos calcular la exponencial de z (es decir, e^z):

```
--> exp(z);
    rectform(%);
```

o el logaritmo principal de z:

```
--> log(z);
    rectform(%);
```

Podemos calcular senos y cosenos de z:

```
--> cos(z);
    rectform(%);
```

```
--> sin(z);
    rectform(%);
```

Si preferimos la notación exponencial, "exponentialize()" escribe todo en términos de exponenciales:

```
--> exponentialize(cos(z));
```

y "demoivre()" utiliza senos y cosenos en la salida en lugar de las exponenciales:

```
--> demoivre(cos(z));
```

2 Ecuaciones.

```
--> /*Primero borramos todas las variables que hemos
    usado anteriormente*/

    remvalue(all);
```

En Maxima, una ecuación es una igualdad entre dos expresiones algebraicas escrita con el símbolo =.

```
--> 3*x^2+2*x+x^3-x^2=4*x^2;
```

Podemos asignarle un nombre para referirnos a esa ecuación:

```
--> mi_ecuacion:3*x^2+2*x+x^3-x^2=4*x^2;
```

lo que nos permite operar con ella:

```
--> mi_ecuacion*4-3*x^3;
```

2.1 Resolución de ecuaciones y sistemas.

Maxima puede resolver los tipos más comunes de ecuaciones y sistemas de ecuaciones algebraicas de forma exacta. Por ejemplo, sabe encontrar las raíces de polinomios de grado bajo (2,3 y 4). En cuanto a polinomios de grado más alto o ecuaciones más complicadas, no siempre será posible encontrar la solución exacta. En este caso, podemos intentar encontrar una solución aproximada en lugar de la solución exacta.

La orden "solve" nos da todas las soluciones, ya sean reales o complejas de una ecuación sistema de ecuaciones:

```
--> solve(x^2-3*x+1=0,x);
```

También podemos resolver ecuaciones que dependan de algún parámetro:

```
--> solve(x^3-a*x^2-x^2+2*x=0,x);
```

Podemos obtener la multiplicidad de las raíces:

```
--> solve(x^7-2*x^6+2*x^5-2*x^4+x^3=0);
```

```
--> multiplicities;
```

La orden solve no sólo puede resolver ecuaciones algebraicas:

```
--> solve(sin(x)*cos(x)=0,x);
```

Como cualquiera puede imaginarse, Maxima no resuelve todo. Incluso en las ecuaciones más “sencillas”, los polinomios, se presenta el primer problema: no hay una fórmula en términos algebraicos para obtener las raíces de un polinomio de grado 5 o más. Pero no hay que ir tan lejos. Cuando añadimos raíces, logaritmos, exponenciales, etc., la resolución de ecuaciones se complica mucho. En esas ocasiones lo más que podemos hacer es ayudar a Maxima a resolverlas.

```
--> eq:x+3=sqrt(x+1);
```

```
--> solve(eq,x);
```

```
--> solve(eq^2);
```

También podemos resolver sistemas de ecuaciones. Sólo tenemos que escribir la lista de ecuaciones y de incógnitas. Por ejemplo:

```
--> solve([x^2+y^2=1,(x-2)^2+(y-1)^2=4],[x,y]);
```

Si la solución depende de un parámetro o varios, Maxima utilizará %r1, %r2,... para refererirse a estos. Por ejemplo,

```
--> solve([x+y+z=3,x-y=z],[x,y,z]);
```

¿Qué pasa si el sistema de ecuaciones no tiene solución? Veamos un ejemplo:

```
--> solve([x+y=0,x+y=1],[x,y]);
```

2.2 Resolución aproximada de ecuaciones polinómicas.

Las ecuaciones polinómicas se pueden resolver de manera aproximada. Los comandos allroots (que nos da todas las raíces) y realroots (que solo nos da las raíces reales) están especializados en encontrar soluciones racionales aproximadas de polinomios en una variable:

```
--> eq:x^9+x^7-x^4+x$
    allroots(eq);
```

y si solo nos interesan las que son reales:

```
--> realroots(eq);
```

3 Análisis Complejo II: Funciones analíticas.

Con wxMaxima podemos calcular directamente algunos límites de funciones complejas. En concreto, vamos a poder resolver límites siempre que la función venga dada en términos de z y la misma sea analítica (ya que en este caso usamos, básicamente, los mismos procesos que en el caso de funciones con una variable).
Así, vamos a poder calcular límites de la forma siguiente (es decir, límites que existen)

EJEMPLO 1: Calcular los siguientes límites:

```
--> limit(z/(z+1), z, 1);
```

```
--> limit((%i*z+3)/(z+1), z, -1);
```

NOTA: Para el caso que no exista el límite, o también siempre que la variable z venga expresada en términos de x+%iy, será más aconsejable expresar la función f(z) en la forma
f(z)=u(x,y)+%iv(x,y)
y calcular (como límites de funciones de dos variables) tanto el límite de u(x,y) como el de v(x,y).
Lo vemos en el siguiente ejemplo:

EJEMPLO 2: Calcular el siguiente límite:

```
--> limit((conjugate(z)+%i*z^2)/cabs(z), z, 0);
```

Observamos que nos da un mensaje de error ya que limit es una función de una sola variable, y en este caso, al llevar la función de la que se quiere calcular su límite tanto un conjugado como un valor absoluto (módulo), ésta no va a ser analítica, por lo que para obtener su límite tendremos que expresarla como dos funciones de dos variables reales (parte real u(x,y) y parte imaginaria v(x,y)):

```
--> z:x+%i*y;
```

```
--> (conjugate(z)+%i*z^2)/cabs(z);
```

```
--> expand(%);
```

```
--> f:rectform(%);
```

y elegimos la parte real u(x,y) y la parte imaginaria v(x,y):

```
--> u:realpart(f);
```

```
--> v:imagpart(f);
```

Entonces, y usando las técnicas de cálculo de límites dobles estudiadas en 1er curso, podremos probar la no existencia de los límites de u(x,y) y/o v(x,y) (a través de límites direccionales o mediante un cambio a polares). También puede verse como hacerlo con wxMaxima en la página 111 del manual que se encuentra en la web:
http://es.scribd.com/doc/52510808/Manual-wxMaxima

Con wxMaxima también podemos estudiar cuando una función compleja es analítica, ya que básicamente consiste en aplicar las ecuaciones de Cauchy-Riemann, que conlleva el cálculo de derivadas parciales de las funciones parte real u(x,y) y parte imaginaria v(x,y):

EJEMPLO 3. Estudiar si la función f(z)=abs(z)^2 admite derivada:

```
--> z:x+%i*y;
    fz: cabs(z)^2;

--> u:realpart(fz);
    v:imagpart(fz);

--> /*calculamos d/dx(u) y d/dy(u)*/
    ux: diff(u,x);
    uy: diff(u,y);

--> /*calculamos d/dx(v) y d/dy(v)*/
    vx: diff(v,x);
    vy: diff(v,y);

--> /* Planteamos las dos ecuaciones de Cauchy Riemann */
    /* Primera d/dx(u) = d/dy(u) */
    ecr1 : ux = vy;
    /* Segunda d/dy(u) = -d/dx(v)*/
    ecr2 : uy = -vx;
    /*Resolvemos el sistema */
    solve([ecr1,ecr2],[x,y]);
```

Observamos que no se verifican las ecuaciones de C-R salvo si x=y=0. Por tanto, f'(z) no existe salvo si z=0.

EJEMPLO 4. Estudiar si la función f(z)=exp(z) admite derivada:

```
--> exp(z);
    rectform(%);

--> u:realpart(exp(z));
    v:imagpart(exp(z));

--> /*calculamos d/dx (u) y d/dy (u)*/
    ux : diff(u,x);
    uy : diff(u,y);

--> /*calculamos d/dx (v) y d/dy (v)*/
    vx: diff(v,x);
    vy: diff(v,y);

--> /* Planteamos las dos ecuaciones de Cauchy Riemann */
    /* Primera d/dx(u) = d/dy(u) */
    ecr1 : ux = vy;
    /* Segunda d/dy(u) = -d/dx(v)*/
    ecr2 : uy = -vx;
    /*Resolvemos el sistema */
    solve([ecr1,ecr2],[x,y]);
```

Observamos que se verifican las ecuaciones de C-R, por lo que f(z) es derivable. Además sabemos que f'(z)= diff(u,x)+%i*diff(v,x):

```
--> derivada_f:diff(u,x)+%i*diff(v,x);
```

Hemos llegado al conocido resultado de que si f(z)=exp(z), entonces f'(z)=f(z). A esta misma conclusión podríamos haber llegado si hubiésemos realizado

```
--> remvalue(all)$
    /*Primero hemos borrado todas las variables antes de calcular
    la derivada tomando z como una variable*/
    diff(exp(z),z);
```

De nuevo en este ejemplo hemos calculado una derivada como si la función a derivar no fuese una función compleja, sino que fuese una función con una sola variable. Esto siempre puede hacerse en caso que la función a derivar sea analítica, mientras que si no lo es (porque, por ejemplo, lleva un módulo o un conjugado en su definición) no lo vamos a poder hacer (como hemos visto en el Ejemplo 3 anterior).

Otro típico ejercicio de variable compleja relacionado con la derivación consiste en dada la parte real u(x,y) de una función analítica, hallar su función armónica conjugada (es decir, otra función v(x,y) tal que la función dada por f(z)=u(x,y)+iv(x,y) es analítica). Lo vemos en el siguiente ejemplo.

EJEMPLO 5. Dada la función u(x,y)=3*x^2*y+2*x^2-y^3-2*y^2, probar que es armónica y hallar su armónica conjugada:

```
--> u(x,y):=3*x^2*y+2*x^2-y^3-2*y^2;
```

```
--> diff(u(x,y),x,2)+diff(u(x,y),y,2);
```

Como este último resultado da 0, es cierto que u(x,y) es armónica. Para hallar la armónica conjugada, como sabemos que d/dy(v)=d/dx(u), entonces v(x,y)=int(d/dx(u),respecto y)+g(x)

```
--> diff(u(x,y),x);
```

```
--> integrate(diff(u(x,y),x), y);
```

Este último resultado nos da el valor para v(x,y), salvo una cte (que es una función que depende de x, g(x)). Usaremos la segunda de las condiciones de Cauchy-Riemann para calcular g(x). Por la segunda de estas ecuaciones se tiene que d/dx(v)=-d/dy(u). Calculamos primero d/dx(v), para lo que derivaremos respecto de x el último resultado obtenido anteriormente:

```
--> diff(integrate(diff(u(x,y),x), y),x,1);
```

Aquí tendremos que sumarle a la expresión anterior g'(x).
El segundo miembro de la igualdad d/dx(v)=-d/dy(u) también lo obtenemos haciendo

```
--> diff(u(x,y),y)*(-1);
```

y para obtener g'(x) hacemos

```
--> diff(u(x,y),y)*(-1)-diff(integrate(diff(u(x,y),x), y),x,1);
```

e integrando este último resultado obtendremos g(x)

```
--> integrate(%, x);
```

Al final resulta que v(x,y)=3*x*y^2+4*x*y-x^3.