Suavizado de curvas

0. Introducción

Cargamos los ejemplos y los métodos auxiliares que ayudan al desarrollo del problema y que se usarán a lo largo del documento.

```
ejemplo1 = load(DATA+"ejemplo1.sobj")
```

```
ejemplo2 = load(DATA+"ejemplo2.sobj")
```

```
ejemplo3 = load(DATA+"ejemplo3.sobj")
```

```
ejemplo4 = load(DATA+"ejemplo4.sobj")
```

```
ejemplo5 = load(DATA+"ejemplo5.sobj")
```

Área de un triángulo

Es de notar que este área puede ser negativa, dependiendo de la posición de los puntos y el giro en el que se calcule el área.

```
def areaTriangulo(a,b,c):
    area = (1/2)*(a[0]*(b[1]-c[1])+b[0]*(c[1]-a[1])+c[0]*(a[1]-
b[1]))
    return area
```

Área de un polígono cualquiera centrado en un punto

El área de un polígono cualquiera se puede calcular obteniendo el área de los triángulos que se forman entre dos pares de vértices del polígono y el punto origen.

```
def areaPol (poligono):
    lista = []
    for i in range(len(poligono)):
       ar = areaTriangulo([0,0],poligono[i],poligono[i-1])
       lista.append(ar)
       i += 1
    res = sum(lista)
```

```
    return res
```

Distancia entre dos puntos

Método que devuelve la distancia usual en $\mathbb{R}$x$\mathbb{R}$ entre dos puntos, es decir, la norma.

```
def distancia(x,y):
    return sqrt((x[0]-y[0])**2 + (x[1]-y[1])**2)
```

Filtrado de puntos

Método que filtra y elimina los datos anómalos de una lista de puntos.

Con datos anómalos nos referimos a datos provenientes de ruido en la muestra o que no son muy cercanos a la mayoría de los puntos.

```
def filtrarPuntos(P, minPuntos):
    lista = []
    i = 0
    umbral = 5 * mean([distancia(P[j-1],P[j]) for j in
range(0,len(P)-1)])
    while i < len(P):
        if distancia(P[i-1], P[i]) <= umbral:
            lista.append(P[i])
            i+=1
        #Si no está en el umbral de distancia, entramos en modo de
limpieza de puntos
        else:
            ptoAnterior = P[i-1]
            contador = 0
            aux = []
            while contador < minPuntos and contador != -1 and i <
len(P):
                if distancia(ptoAnterior, P[i]) <= umbral:
                    #Hay datos anómalos. Ignoramos los anteriores y
seguimos.
                    del aux[:]
                    aux.append(P[i])
                    contador = -1
                else:
                    #Pueden ser puntos muy alejados, los guardamos
por si acaso.
                    aux.append(P[i])
                    contador += 1
                i+=1
```

```
            for pto in aux:
                lista.append(pto)
    return lista
```

Mostrar polígono

El método muestra en una gráfica la curva poligonal a la que se le ha aplicado el algoritmo "funcion" un número "veces" de iteraciones, junto con el polígono original.

```
def showPolygon(funcion, puntos, veces):
    E = deepcopy(puntos)
    for i in range(veces):
        E = funcion(E)
    return line(E+[E[0]],aspect_ratio=1,color="red") + line(puntos)
```

Criterio de buen suavizado según las áreas

Uno de los criterios que se han utilizado para evaluar el “grado de suavizado" de una curva poligonal ha sido la comparación de áreas. Para considerar que una curva está bien suavizada, calculamos la diferencia entre el área de la figura original y el área de la poligonal resultante.

```
def buenSuavizado(funcion, ejemplo, pasos):
    ej = deepcopy(ejemplo)
    for i in range(pasos):
        ej = funcion(ej)
    areaSuvEj = areaPol(ej)
    areaIni = areaPol(ejemplo)
    suv = abs(areaSuvEj - areaIni)
    coc = suv/100
    if (coc < 0.1):
        res = "Buen suavizado"
    else:
        res ="No buen suavizado"
    return res
```

1. Suavizado de curvas poligonales

Antes necesitamos definir el tipo de curvas que se van a estudiar: las curvas poligonales son varios segmentos de rectas unidos, pueden ser abiertas o cerradas, aunque nosotros trabajaremos con ejemplos en los que las curvas son cerradas. Una curva poligonal es abierta cuando los extremos no coinciden en el mismo punto y una línea poligonal es cerrada cuando los extremos sí coinciden en el mismo punto.

El problema de suavizado de curvas poligonales se plantea principalmente para facilitar el estudio y

eliminar los detalles innecesarios y los bucles de curvas poligonales. Las múltiples irregularidades de las curvas poligonales que se nos presentan sin suavizar pueden dificultar el estudio de las mismas, por lo que se plantea este problema en función de agregar facilidad a la hora de tratar con curvas poligonales en los que no son necesarios los detalles, sino una visión más general.

Por ejemplo, centrándonos en el plano geográfico, los cartógrafos desarrollan líneas a mano con una apariencia fluida suave. Sin embargo, a la hora de tratar los datos a través de un ordenador, cuya precisión es limitada, se puede observar que al obtener una imagen más detallada de la curva analizada, se muestra como una serie de segmentos rectos unidos entre sí, muy lejos de la "curvatura" de la curva original (véase en la figura 1).

Comparando los dos elementos de la imagen, se observa que en la curva poligonal "anterior" aparecen aristas rectas, picos y otros signos que impiden su correcta aparición; a diferencia con el suavizado desarrollado "después", donde se tiende a reducir el detalle es cierto, pero los puntos que salen en la figura conforman una única línea, de manera que se borren las pequeñas perturbaciones y se capturen los datos más significativos. De esta manera se puede obtener una visión más generalizada y cómoda para la vista.

Por tanto, uno de los principales usos del suavizado de curvas es mejorar la apariencia y la estética de las representaciones originales de una curva, que la precisión del ordenador ha transformado en poligonal, y que presentan más y mejorescaracterísticas por tanto.

A continuación, repasamos algunos métodos que se han implementado para suavizar una curva poligonal:

MÉTODO DE LOS PUNTOS MEDIOS

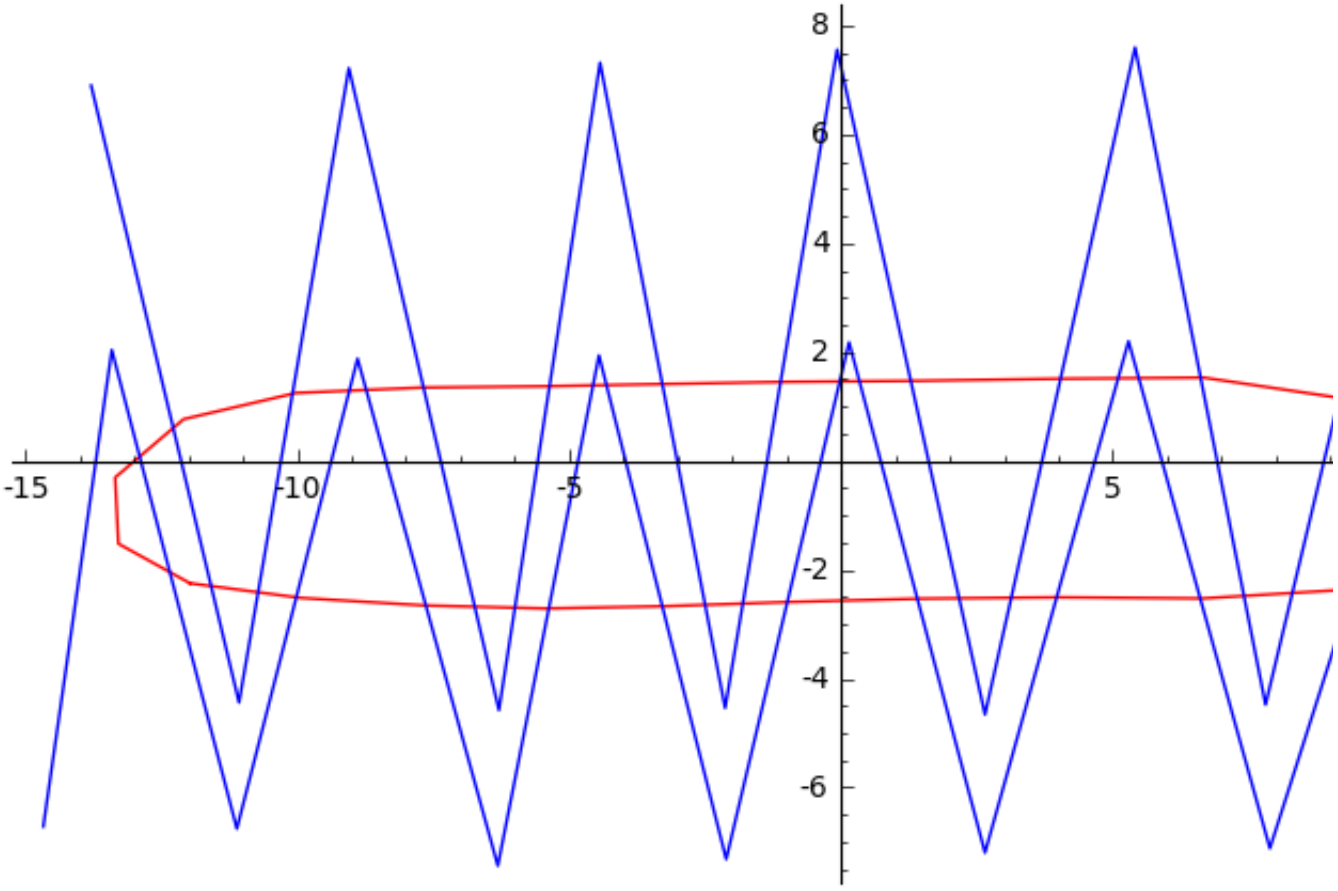
Dada una serie de puntos que unidos forman un polígono, por cada dos vértices conexos/más cercanos cogemos el punto medio entre los vértices de las aristas. Ese punto medio se añade a la nueva lista de puntos de la curva suavizada.

```
def puntosMedios(p):
    lista = []
    for i in range(len(p)):
        der = (p[i-1][1]+p[i][1])/2
        izq = (p[i-1][0]+p[i][0])/2
        lista.append([izq,der])
    return lista
```

Cuantas más iteraciones se realicen de este método, más "suave" quedará la curva resultado. Sin embargo, a la vez la figura cada vez se hará más pequeña. Esto se debe a que los puntos medios cada vez estarán más cerca del centro del polígono y por lo tanto el tamaño del polígono se irá reduciendo.

```
def showPuntosMediosPolygon(polygon, steps):
    E = deepcopy(polygon)
    for i in range(steps):
        E=puntosMedios(E)
    return line(E+[E[0]],aspect_ratio=1,color="red") +
line(polygon)
```

```
showPolygon(puntosMedios, ejemplo1, 5)
```

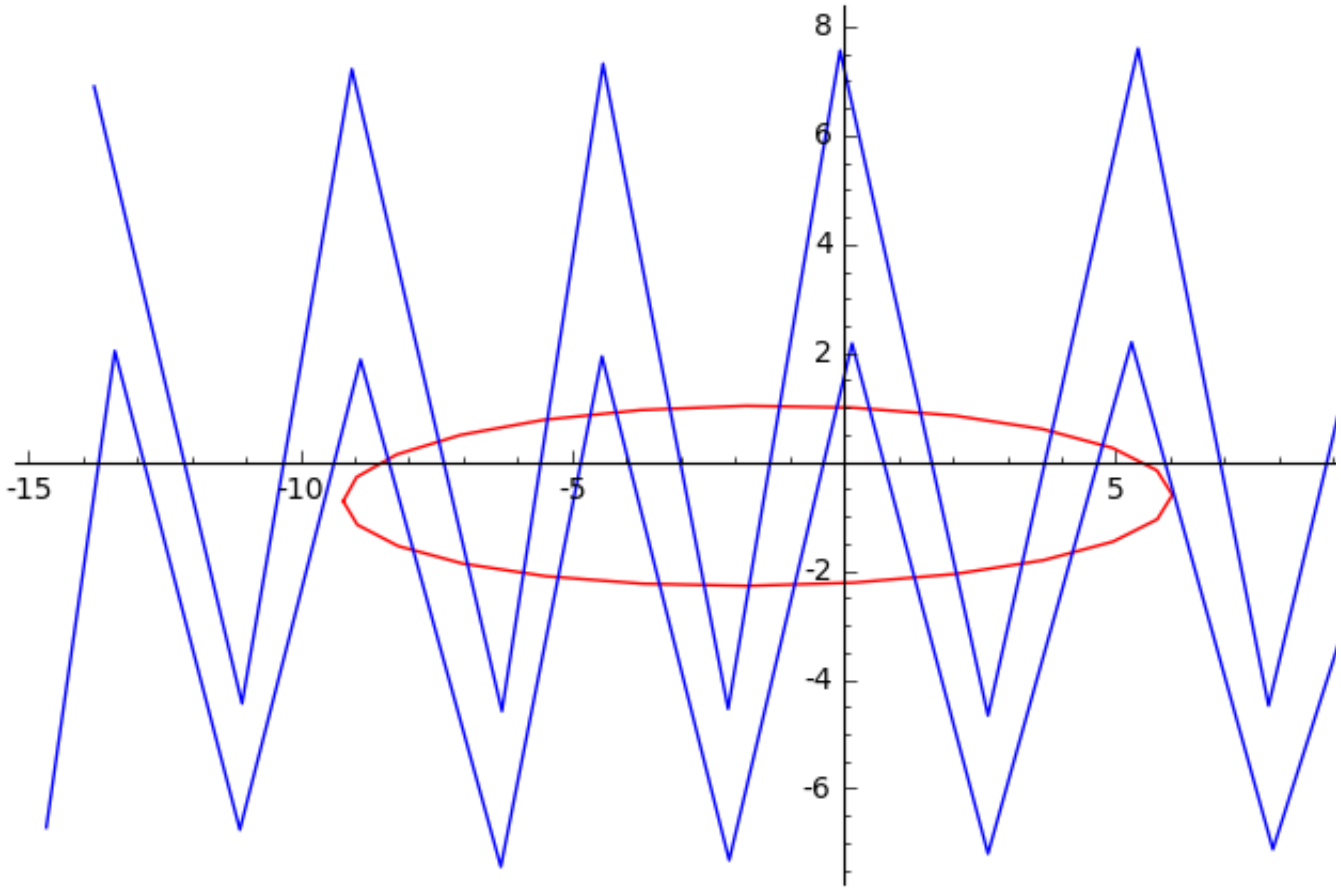



```
buenSuavizado(puntosMedios, ejemplo1, 5)
```

```
'No buen suavizado'
```

Ejecutamos el algoritmo con 5 iteraciones. Observamos que todavía se nota que la curva resultante en realidad son lineas rectas conectadas, pero empieza a tomar la forma de la figura suavizada.

```
showPolygon(puntosMedios,ejemplo1, 50)
```

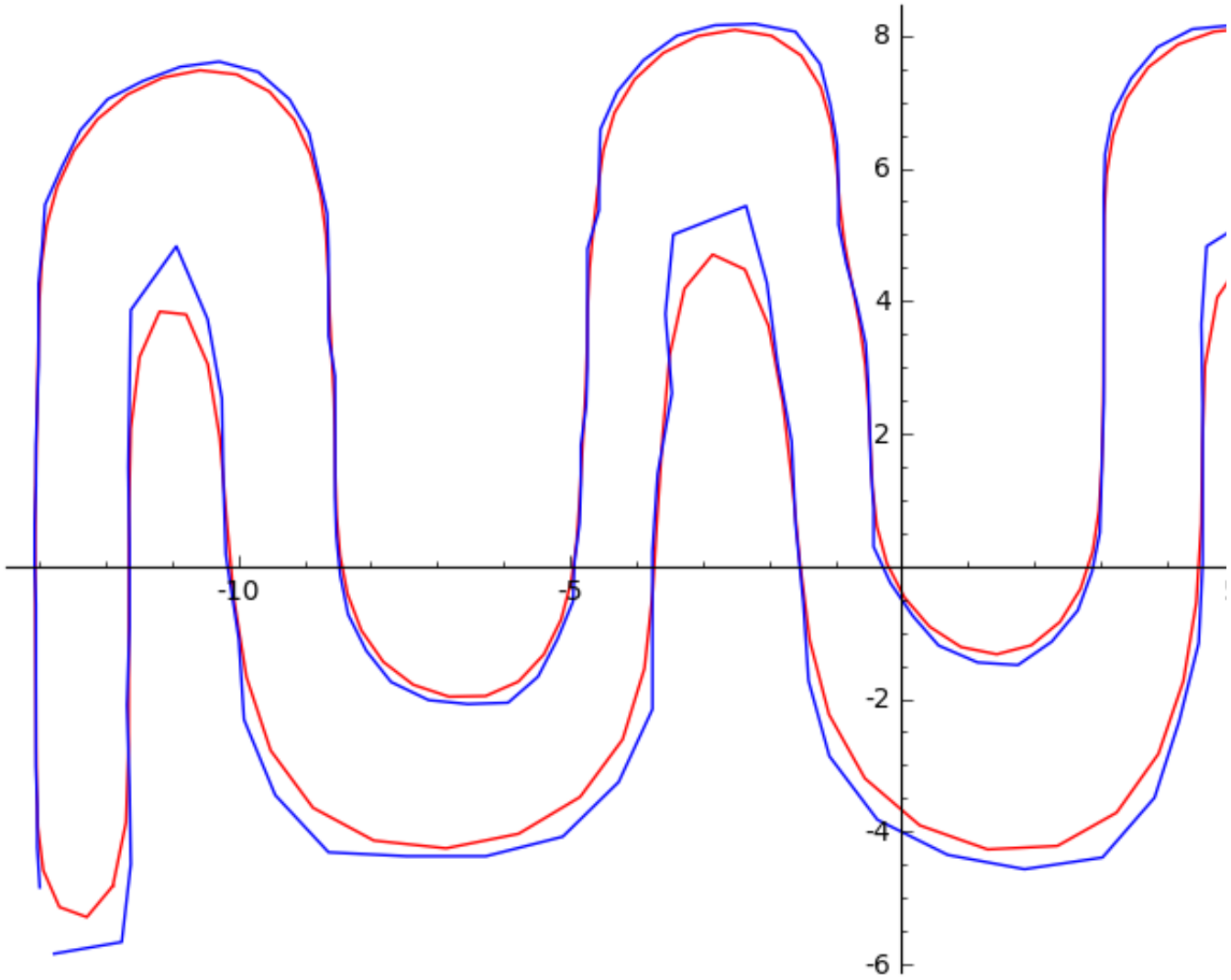



```
buenSuavizado(puntosMedios, ejemplo1, 50)
```

```
'No buen suavizado'
```

Ejecutamos el algoritmo con 50 iteraciones. La figura está muy cercana a la forma de una elipse como una única línea. Sin embargo, la figura suavizada se ha hecho mucho más pequeño que la original.

```
showPolygon(puntosMedios,ejemplo2, 5)
```

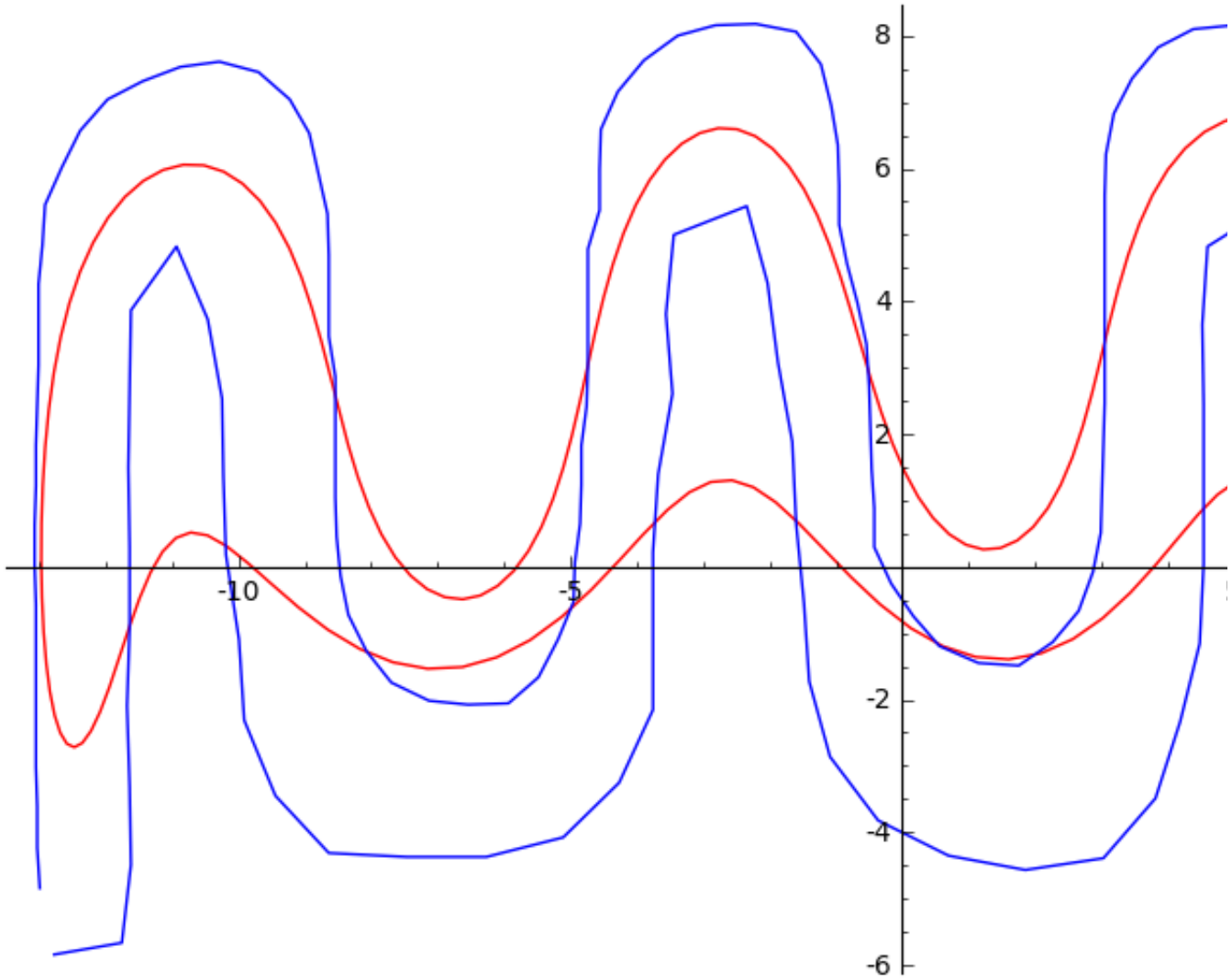



```
buenSuavizado(puntosMedios, ejemplo2, 5)
```

```
'Buen suavizado'
```

Ejecutamos el algoritmo con pocas iteraciones. La figura suavizada se parece mucho a la original en zonas donde la longitud de la normal de dos puntos adyacentes es cercana a cero. En otros puntos, se aleja de la figura original.

```
showPolygon(puntosMedios,ejemplo2, 100)
```

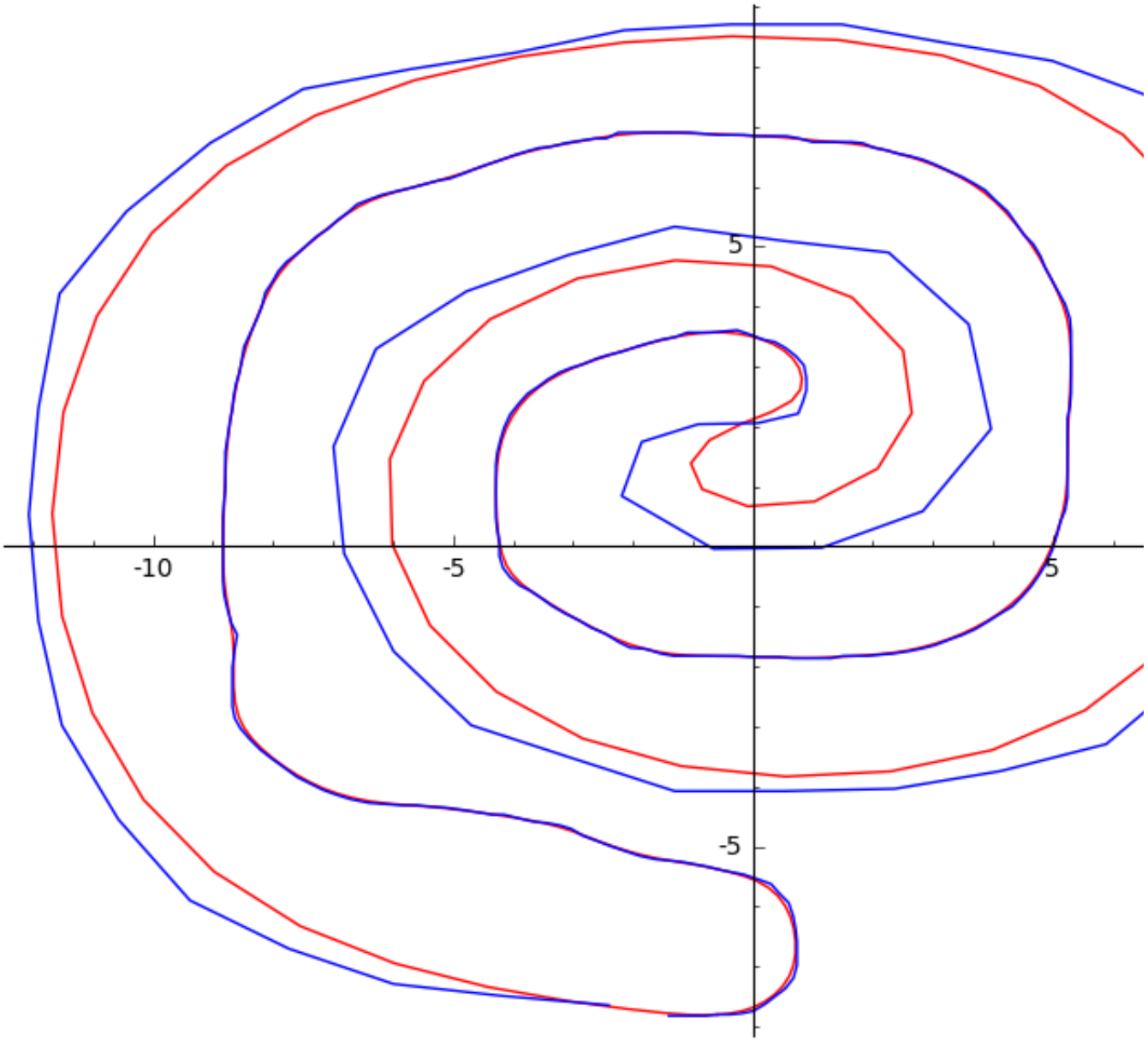



```
buenSuavizado(puntosMedios, ejemplo2, 100)
```

```
'No buen suavizado'
```

Sin embargo en polígonos generados con un número más grande de iteraciones, vemos que el suavizado deja una figura parecida a la original, pero que no se acerca a las líneas que formaban los puntos en la original. La exactitud de la figura se ve en decremento en favor de la suavidad de la nueva curva.

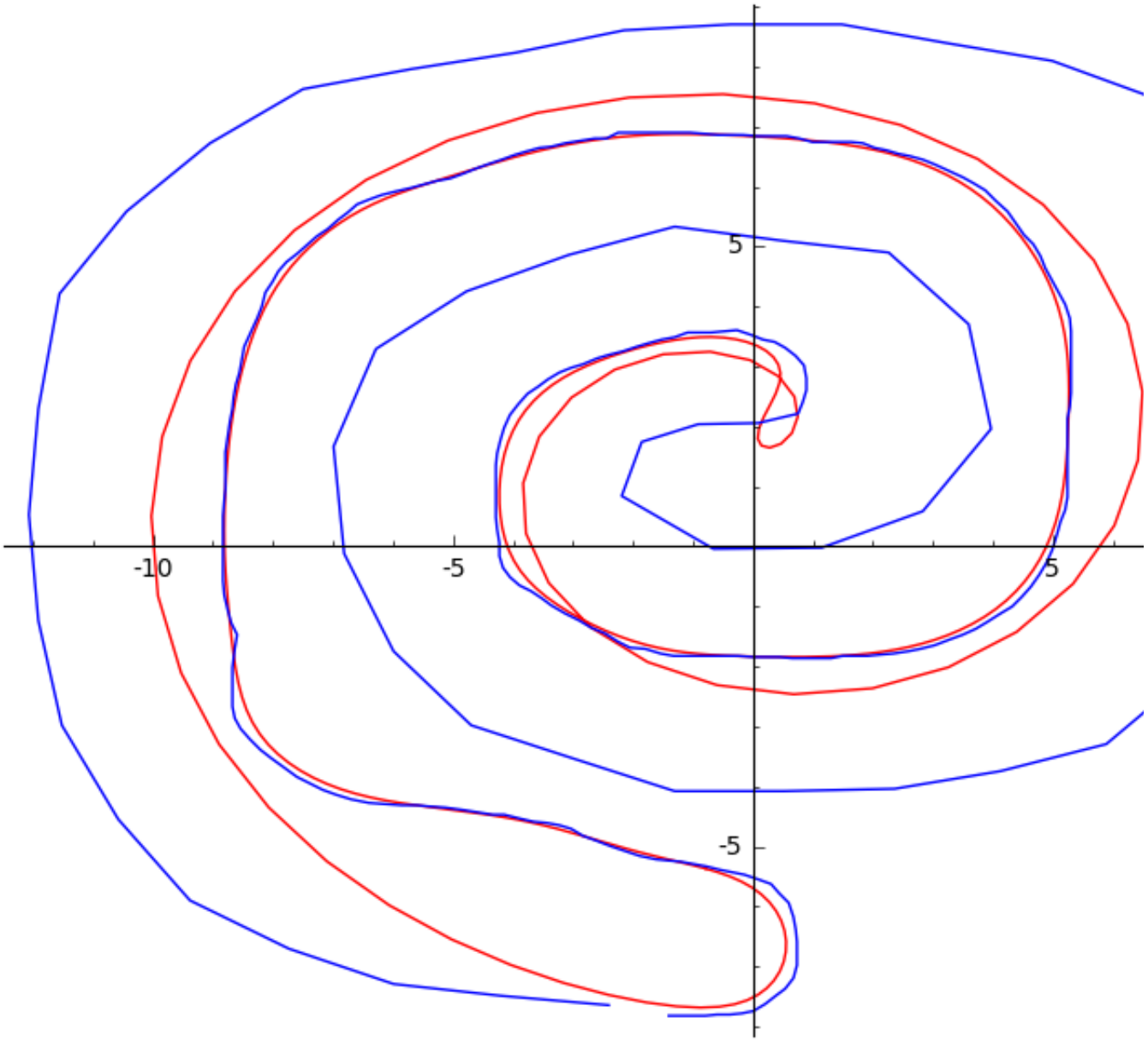
```
showPolygon(puntosMedios,ejemplo3, 10)
```




```
buenSuavizado(puntosMedios, ejemplo3, 10)
```

```
    'No buen suavizado'
```

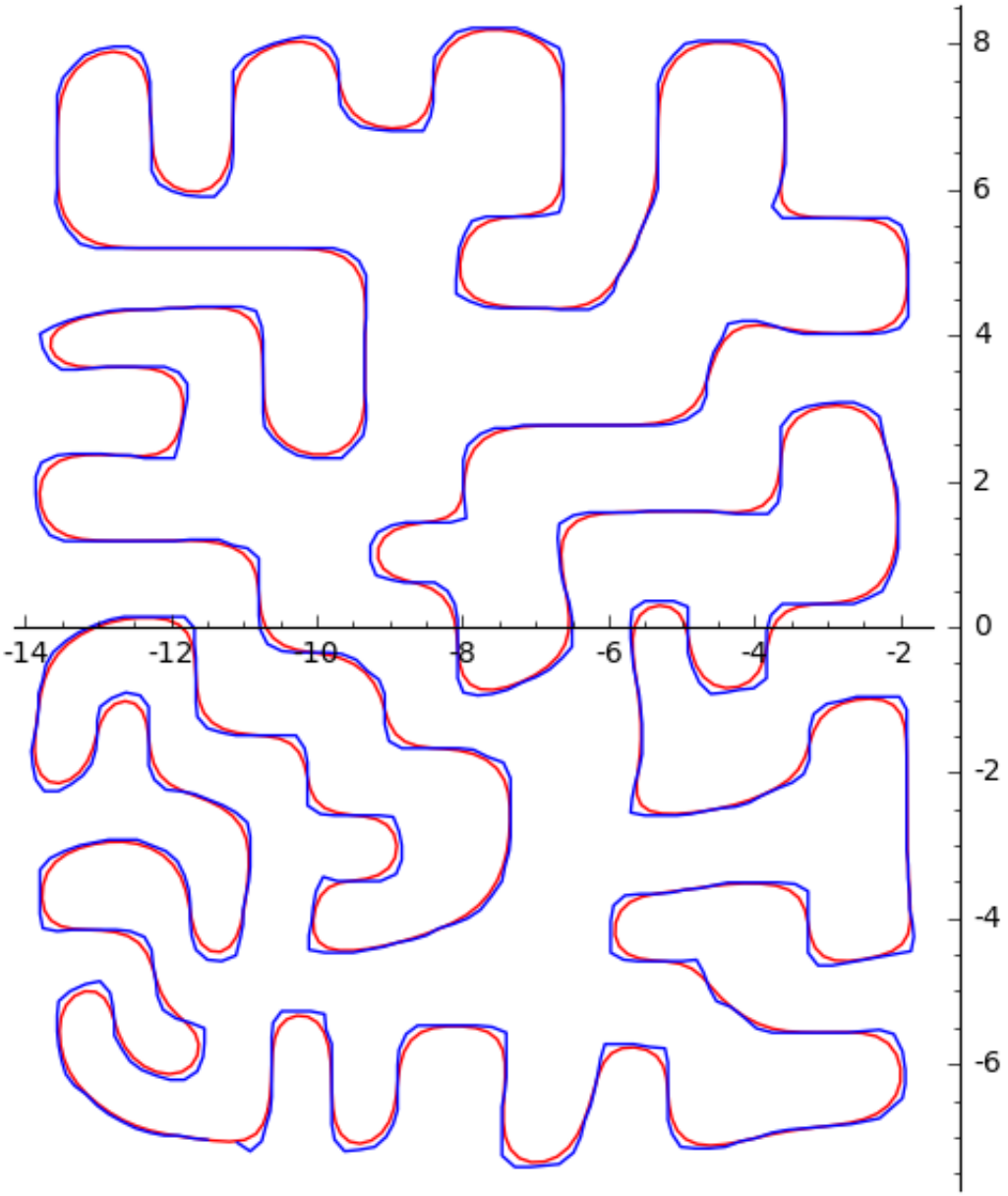
```
showPolygon(puntosMedios,ejemplo3, 50)
```




```
buenSuavizado(puntosMedios, ejemplo3, 50)
```

```
    'No buen suavizado'
```

```
showPolygon(puntosMedios,ejemplo4, 10)
```

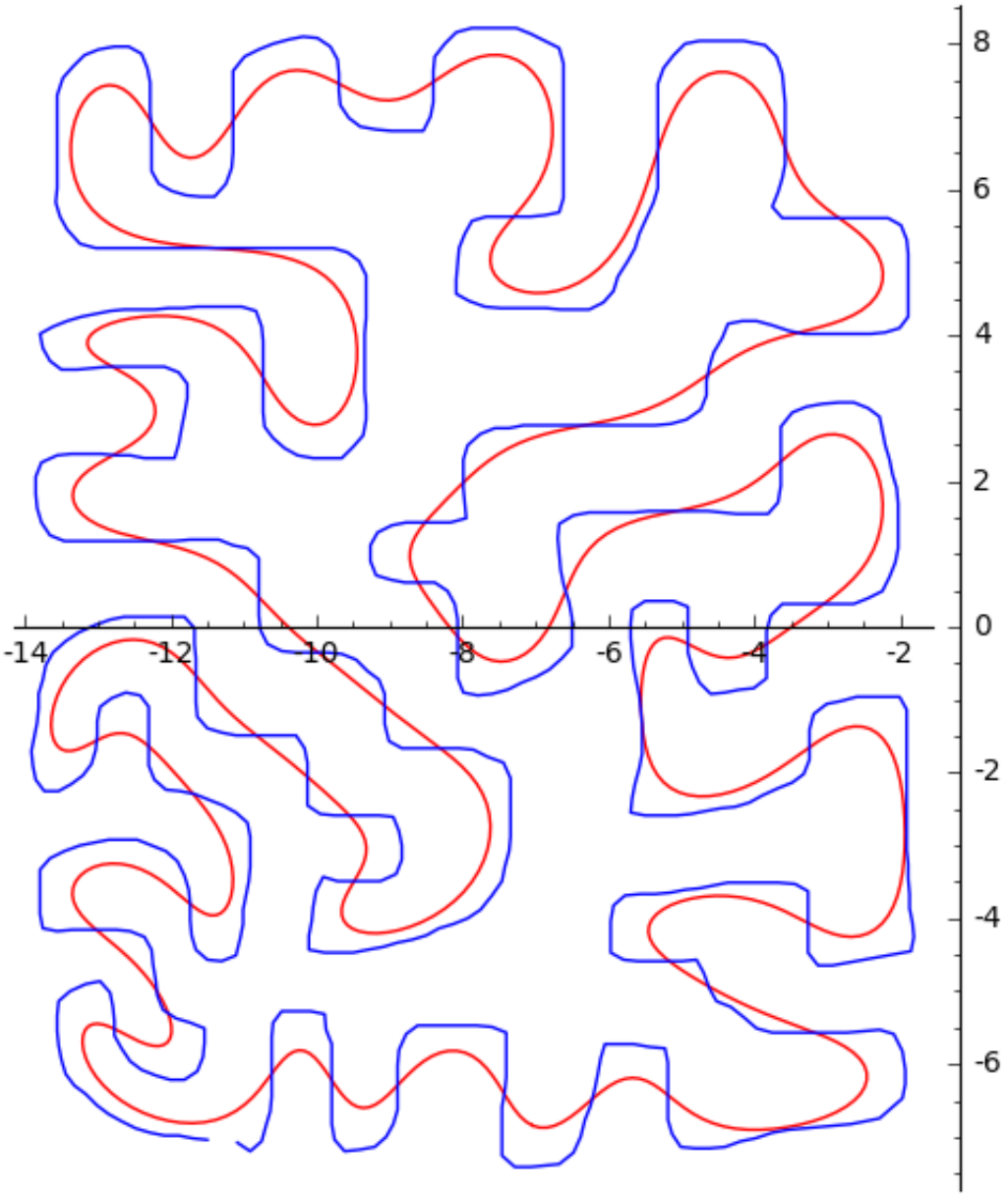



```
buenSuavizado(puntosMedios, ejemplo4, 10)
```

```
'Buen suavizado'
```

En este ejemplo se muestra el algoritmo con pocas iteraciones, que demuestra ser muy eficaz. Se puede observar que la figura resultante es muy parecida a la original, y la curvatura de la misma es bastante eficaz.

```
showPolygon(puntosMedios,ejemplo4, 100)
```

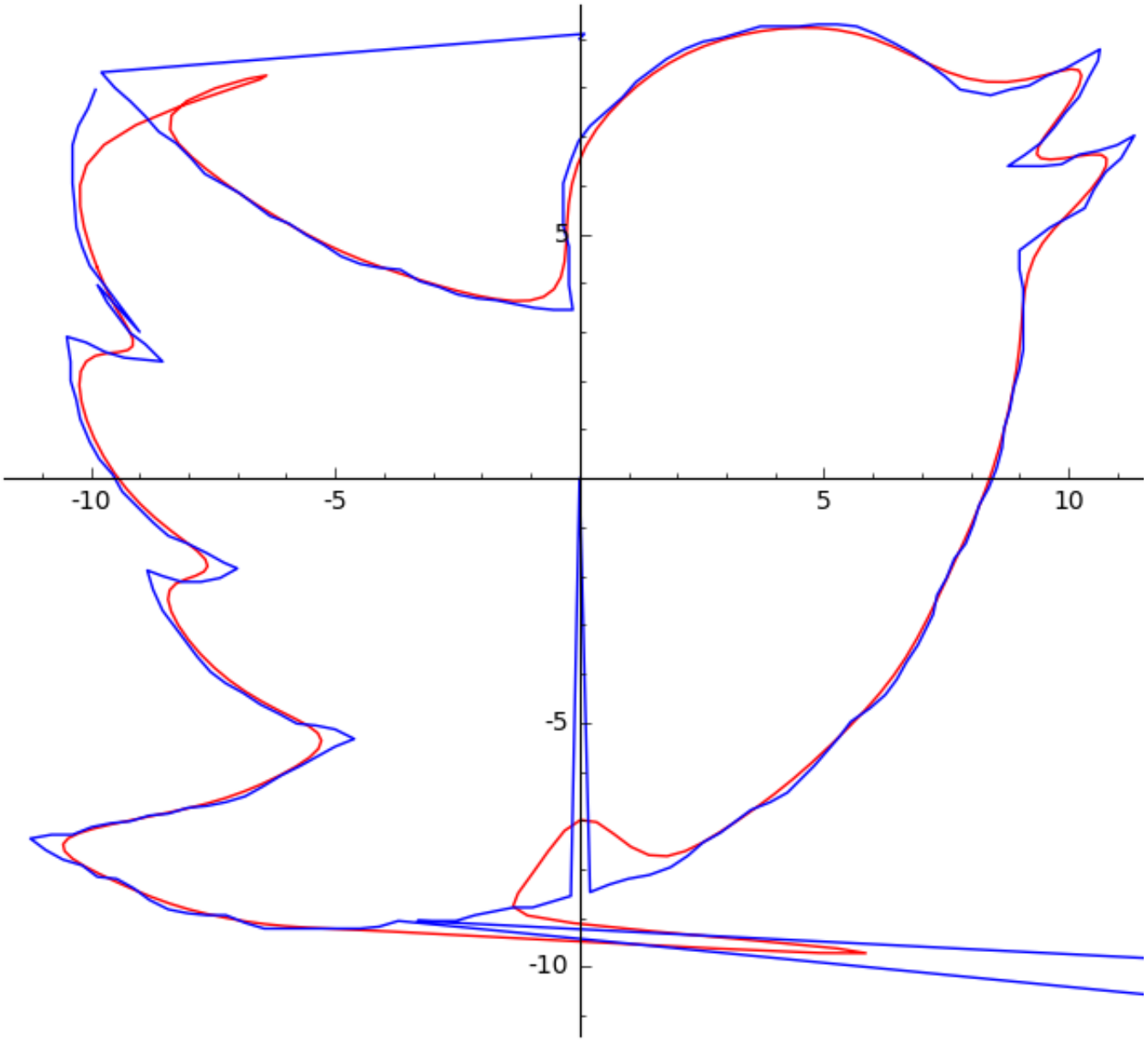



```
buenSuavizado(puntosMedios, ejemplo4, 100)
```

```
'Buen suavizado'
```

Al haber más iteraciones, se puede ver que la figura es menos parecida a la original. Además en sitios con pequeñas variaciones, zonas parecidas a datos anómalos, entonces el algoritmo lo ignora. Esto puede ser bueno para figuras bien marcadas con datos anómalos cercanos a la figura, pero no serían tan eficaz para figuras que requieren mucha precisión.

```
showPolygon(puntosMedios,ejemplo5, 20)
```

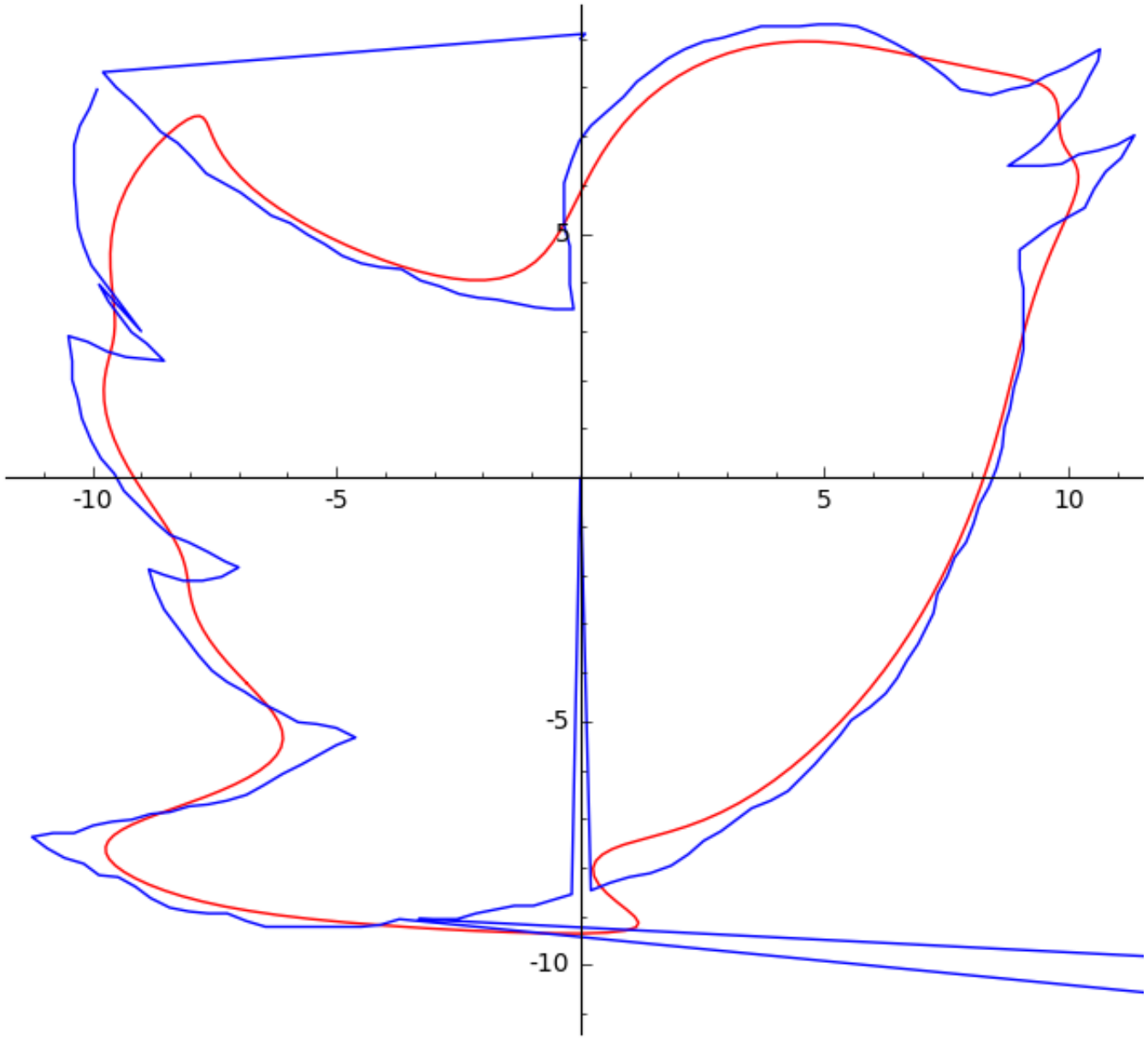



```
buenSuavizado(puntosMedios, ejemplo5, 20)
```

```
'Buen suavizado'
```

Este método con pocas iteraciones se acerca a la figura original en los puntos cuya normal juntos es muy pequeña. Sin embargo, en cambios bruscos de la figura, vemos como la curva poligonalse aleja de la versión suavizada.

```
showPolygon(puntosMedios,ejemplo5, 100)
```



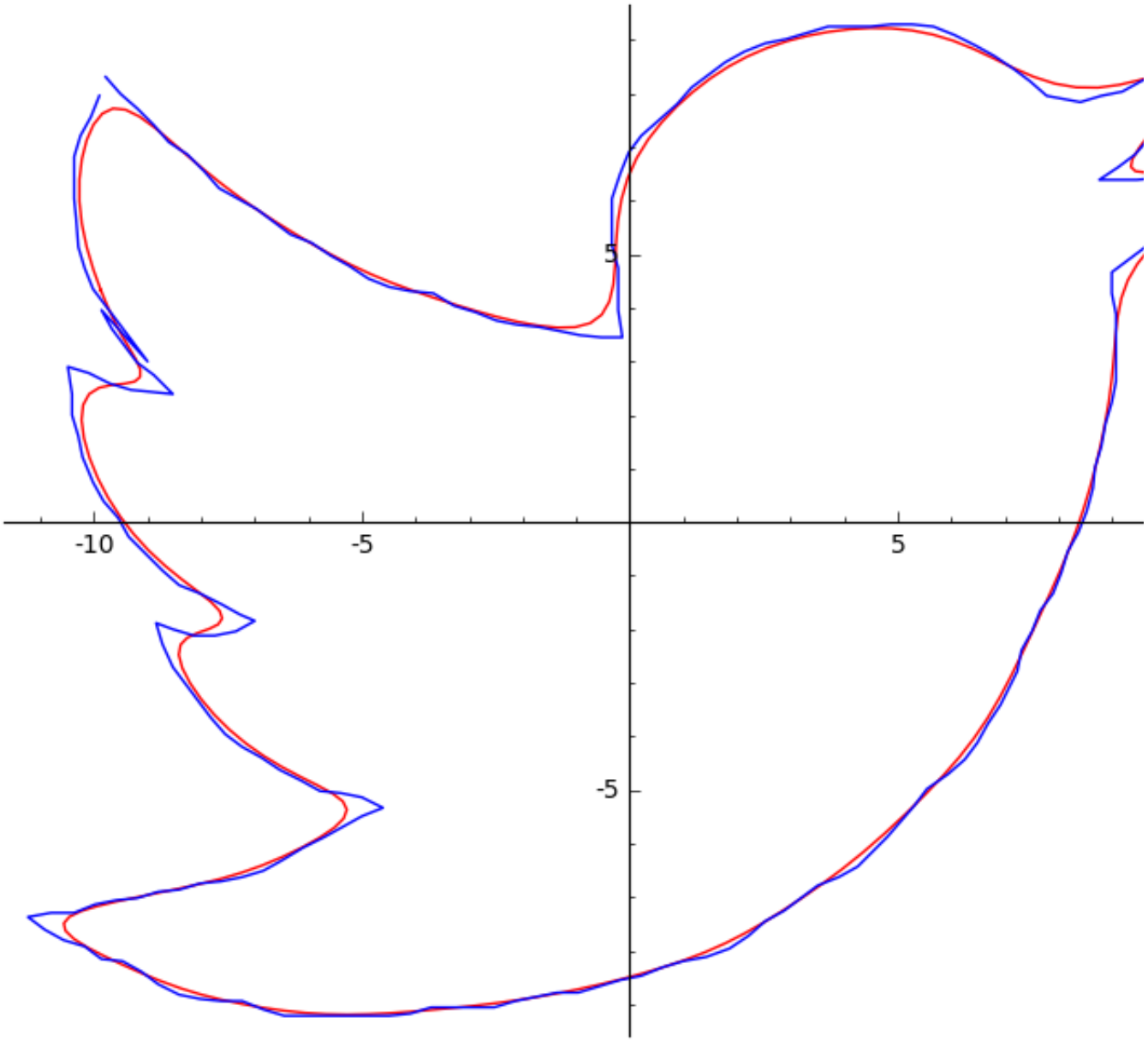

```
buenSuavizado(puntosMedios, ejemplo5, 100)
```

```
'No buen suavizado'
```

Aumentando el número de iteraciones, se observa que la figura se parece cada vez menos a la original. Además ha conseguido salvar de mejor manera los datos anómalos, pero para el número de iteraciones de manera muy ineficiente.

En este ejemplo se presentan datos anómalos que rompen el suavizado esperado de la curva. Se muestra este mismo ejemplo pasado un algoritmo de filtrado de datos anómalos:

```
fixedEjemplo5=filtrarPuntos(ejemplo5,100)
showPolygon(puntosMedios,fixedEjemplo5, 20)
```




```
buenSuavizado(puntosMedios, fixedEjemplo5, 20)
```

```
'Buen suavizado'
```

MÉTODO DE LA ECUACIÓN DEL CALOR DISCRETA

La ecuación del calor es una ecuación diferencial parcial muy importante en matemáticas y física. La ecuación del calor describe la distribución de calor en una determinada región durante un tiempo determinado. La función **u(x,t)** es la que expresa el calor en un punto x durante un tiempo medio uniforme t.

La ecuación del calor (véase figura 5.4) describe la evolución temporal de la temperatura en cada punto x. Como el calor tiende a equilibrarse con el tiempo, la ecuación describe cierto proceso de suavizado, y realizando el promedio el calor u(x) en un punto particular x es afectado por el calor en lugares cercano u(x-β) y u(x+β).

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad (5.4)$$

Con un ejemplo se puede representar el proceso que está sucediendo:

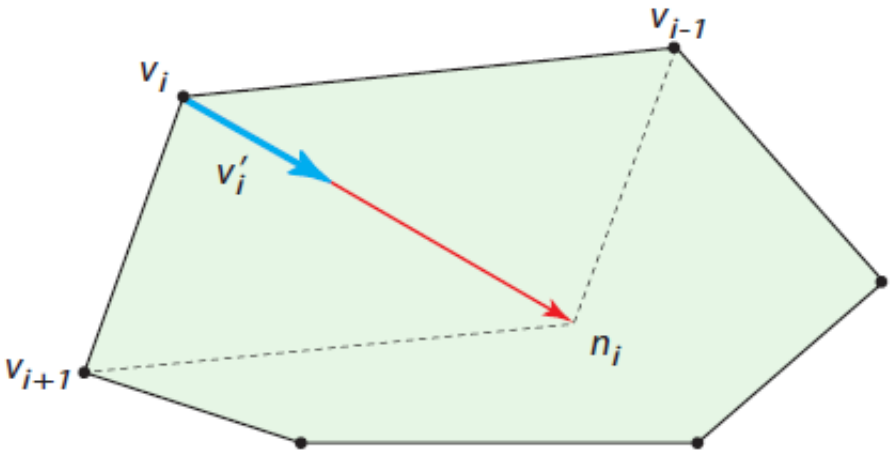
Si por ejemplo, u(x) es un punto más frío que u(x-β) y u(x+β), que son más calientes pero con la misma cantidad, entonces se cancelan los efectos en u(x). Pero si la media de la temperatura, que es 1/2 [u (x-β) + u (x +β)], difiere de u(x), por tanto "tira" de la temperatura en u(x) con una tirón proporcional a la diferencia:

$$u(x) - \frac{1}{2}[u(x-\delta) + u(x+\delta)] = \frac{1}{2}\Big([u(x) - u(x-\delta)] - [u(x+\delta) - u(x)]\Big).$$

Esta expresión es la segunda derivada con respecto de x. No se intenta realizar derivaciones precisas, si no que se pretende hacer una similitud simbólica formal entre la curva de acortamiento de ecuaciones y la ecuación del calor.

Por lo que para seguir con la explicación de la ecuación del calor se debe explicar brevemente la **curva de acortamiento de ecuaciones**, que es la siguiente:

$$\frac{dv_i}{dt} = n_i. \qquad (5.3)$$



Describe la evolución continua de cada vértice (v_{i-1}, v_i, v_{i+1}) discreto del polígono hacia el vector normal.

Después de esta explicación se puede continuar desarrollando la ecuación del calor.

Sustituyendo x en la ecuación (5.4) por s y u(x,t) por C(s,t), aparece una ecuación nueva (5.1).

$$\frac{\partial C}{\partial t} = \frac{\partial^2 C}{\partial s^2} = \kappa n. \qquad (5.1)$$

Esta nueva ecuación estipula que cada punto p de la curva que se tiene se mueve a lo largo de la norma en la curva p con la velocidad que proporciona la curvatura. Esta ecuación describe una figura geométrica fluida y una evolución de la geometría de C a lo largo del tiempo t.

Cambiando la ecuación por la (5.3), se puede pensar en la derivada en el punto medio vi* de vi y vi+1 como aproximación por

$$\frac{\partial v_i^*}{\partial s} \approx v_{i+1} - v_i.$$

Suponiendo esto, se puede aproximar la segunda derivada como:

$$\frac{\partial^2 v_i}{\partial s^2} \approx \frac{\partial v_i^* - \partial v_{i-1}^*}{\partial s} \approx (v_{i+1} - v_i) - (v_i - v_{i-1}).$$

Como el lado derecho es precisamente la ecuación (5.2), podemos ver ecuación (5.3) como

$$\frac{\partial v_i}{\partial t} = \frac{\partial^2 v_i}{\partial s^2}.$$

Además se han realizado ciertos cambios de símbolos y empleado aproximaciones sobre la curva de acortamiento fluido, el fluido discreto... Por tanto, finalmente hemos obtenido la ecuación del calor discreta.

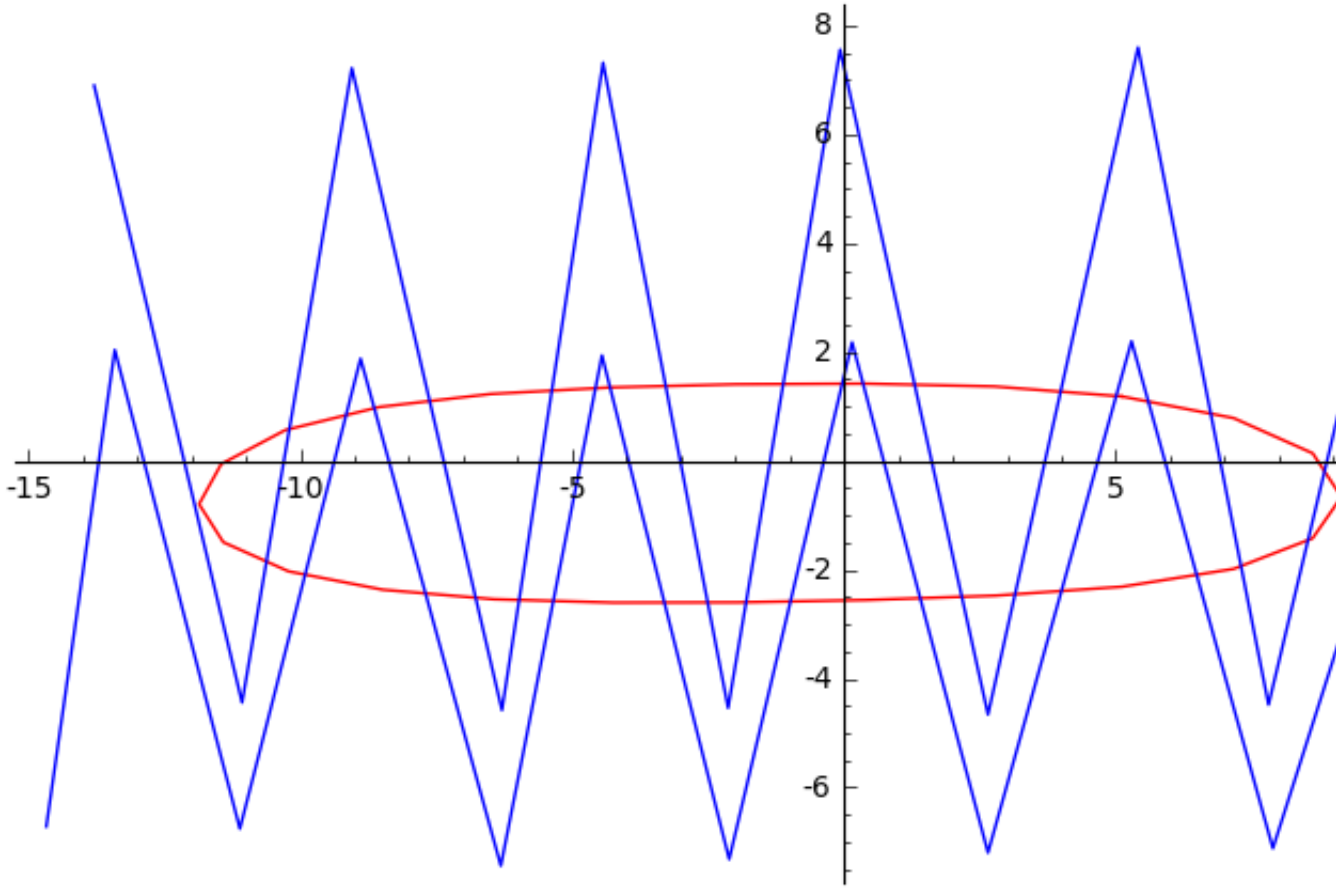
Alfa es el factor multiplicador que empleamos en la ecuación del calor y lo calculamos para obtener un valor fijo que genere las curvas poligonales suavizadas.

```
def calcularAlfa(puntos):
    maximoFinal = 0
    for i in range(len(puntos)):
        if (puntos[i][0]>maximoFinal):
            maximoFinal = puntos[i][0]
        if(puntos[i][1]>maximoFinal):
            maximoFinal = puntos[i][1]
    return maximoFinal
```

```
def ecuacionCalor(p):
    lista = []
```

```
    i = -1
    d = calcularAlfa(p)/100
    while i != len(p)-1:
        niDer = p[i][1]-p[i-1][1]+p[i-2][1]-p[i-1][1]
        niIzq = p[i][0]-p[i-1][0]+p[i-2][0]-p[i-1][0]
        vect = [p[i-1][0] + d*niIzq, p[i-1][1] + d*niDer]
        lista.append(vect)
        i += 1
    return lista
```

```
showPolygon(ecuacionCalor, ejemplo1, 20)
```

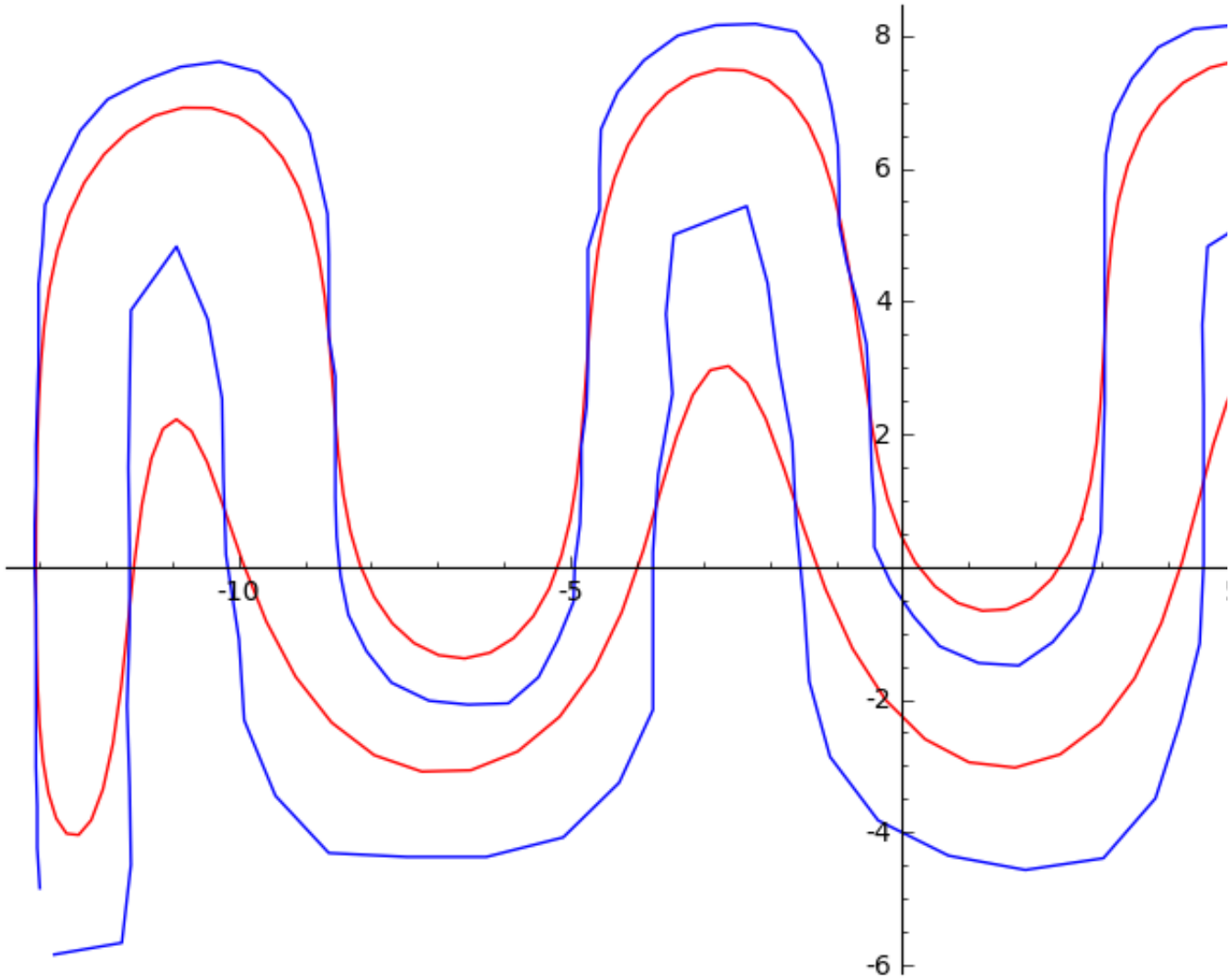



```
buenSuavizado(ecuacionCalor, ejemplo1, 20)
```

```
'No buen suavizado'
```

Se observa que esta curva poligonal posee numerosos picos y por tanto el suavizado que presenta es una circunferencia.

```
showPolygon(ecuacionCalor, ejemplo2, 50)
```

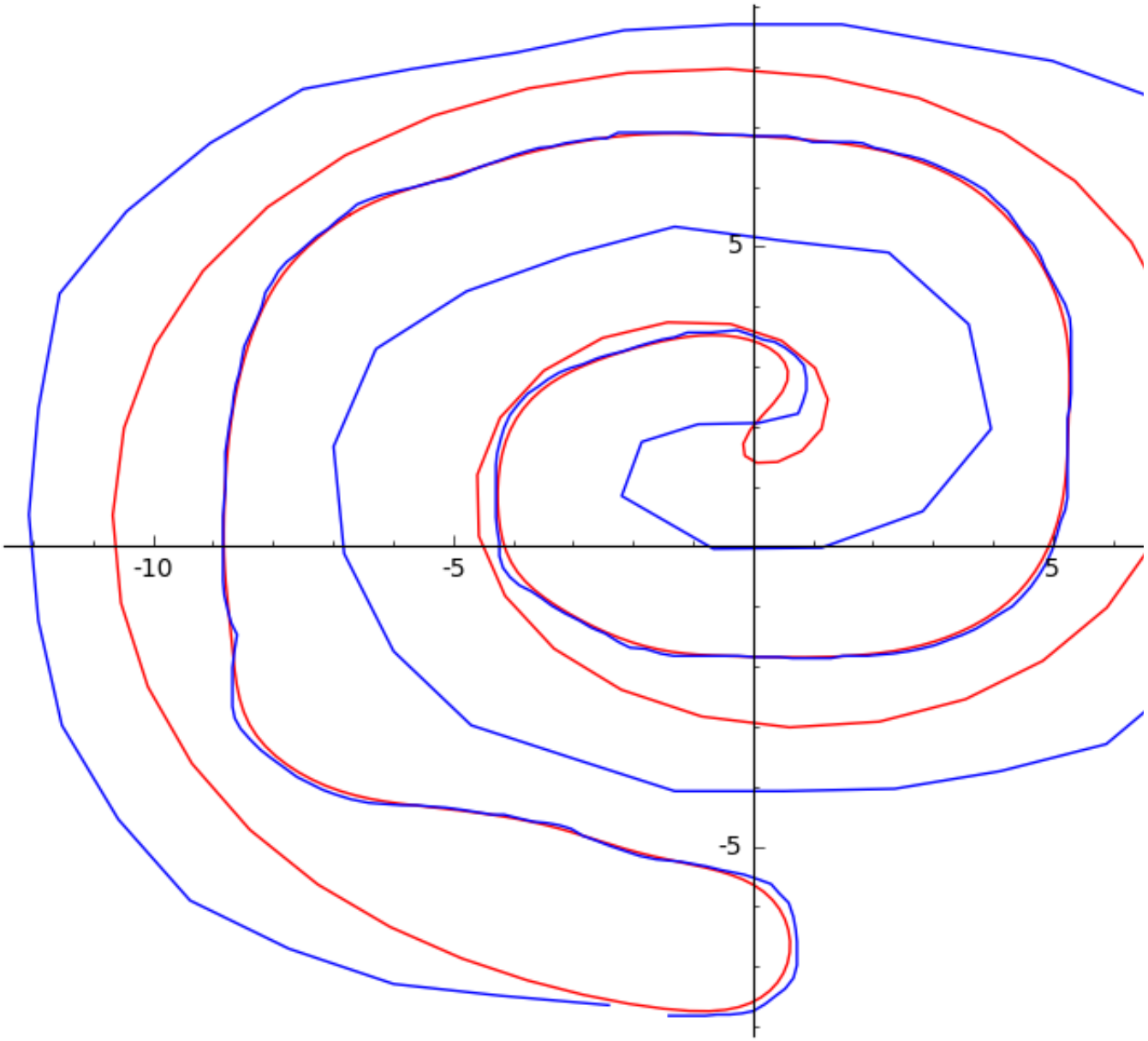



```
buenSuavizado(ecuacionCalor, ejemplo2, 50)
```

```
'No buen suavizado'
```

Esta curva poligonal y las que se muestran a continuación, presentan una forma menos picuda y cuando la suavizamos obtenemos un resultado muy cercano al deseado.

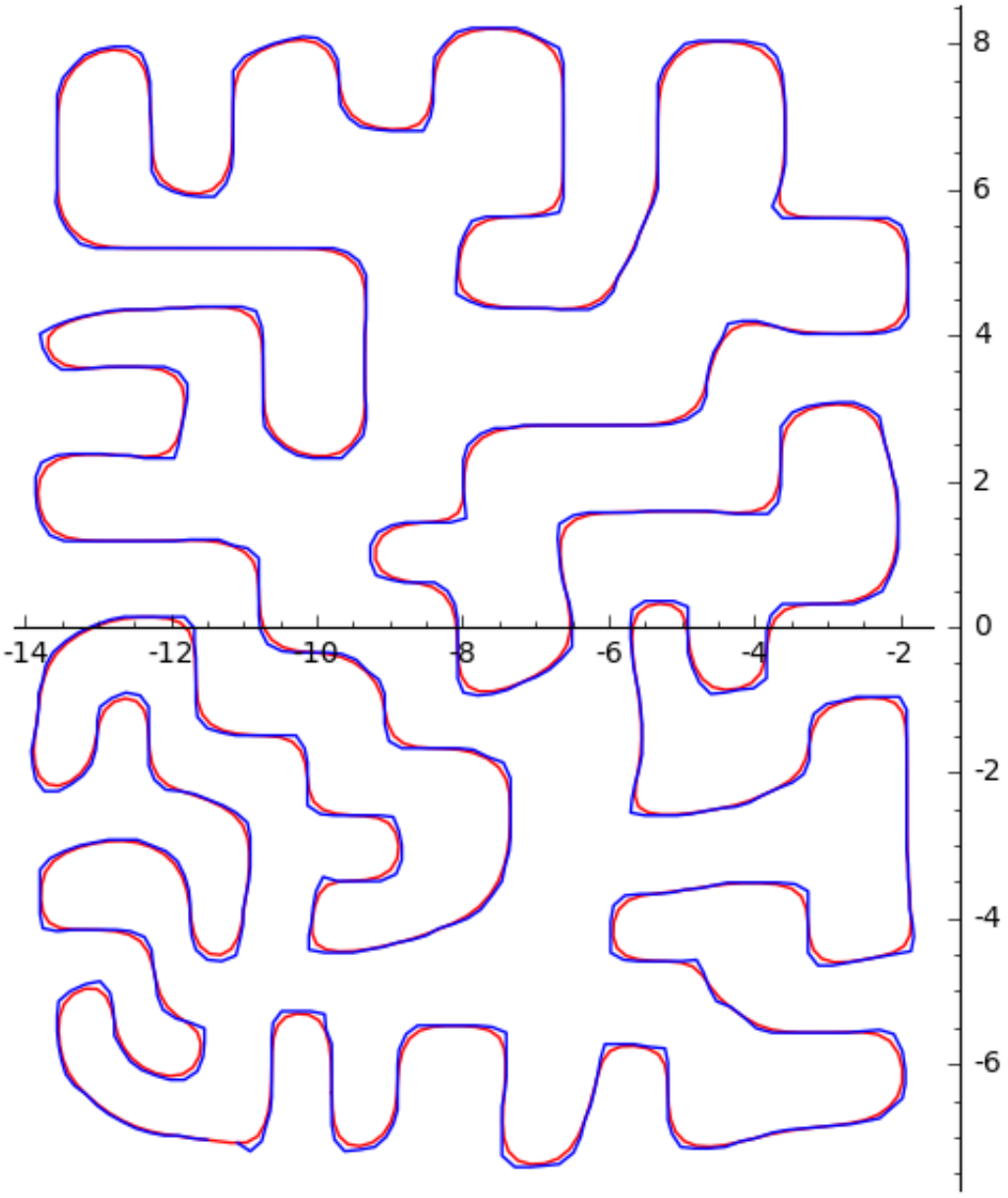
```
showPolygon(ecuacionCalor, ejemplo3, 50)
```




```
buenSuavizado(ecuacionCalor, ejemplo3, 50)
```

```
    'No buen suavizado'
```

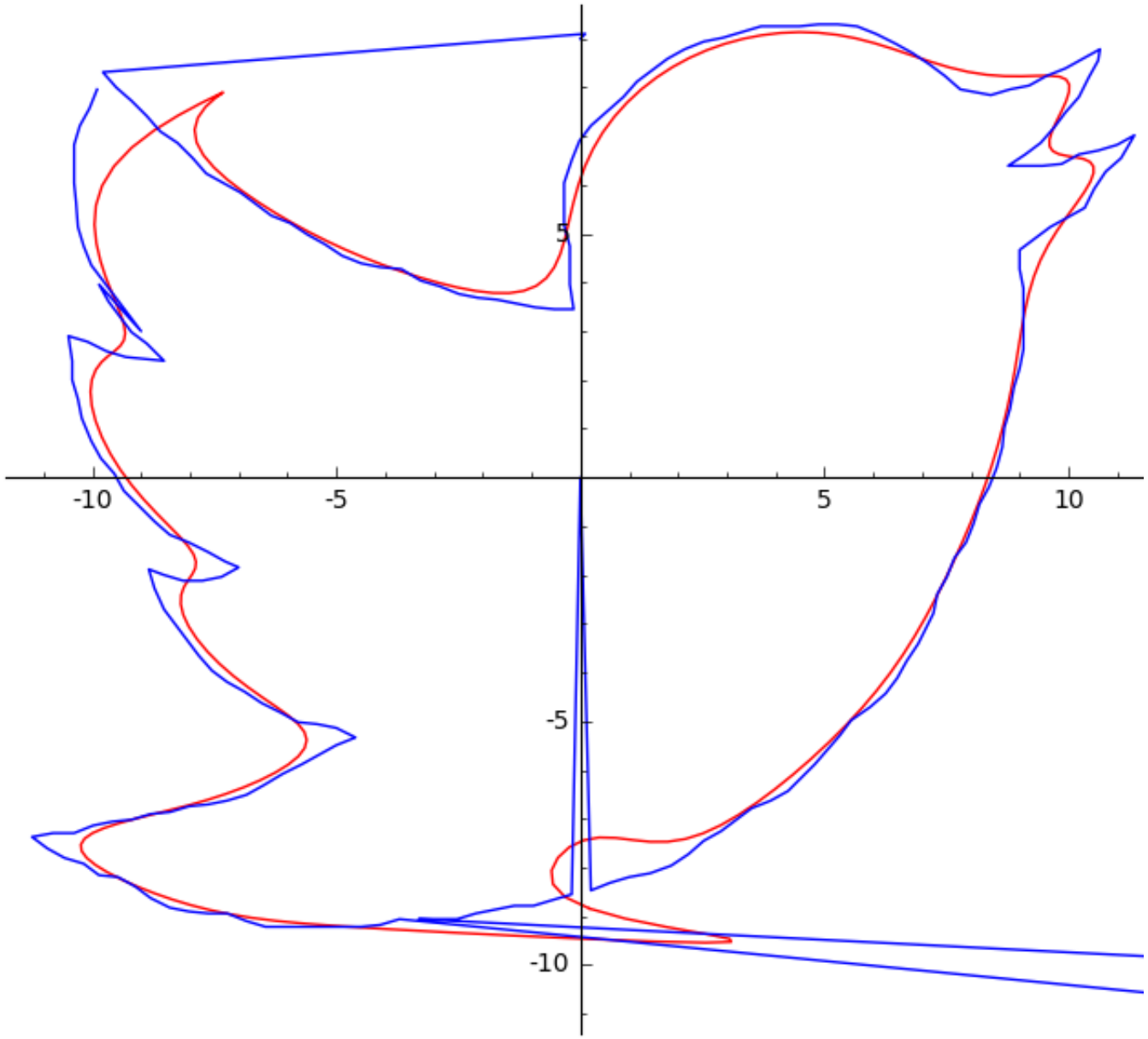
```
showPolygon(ecuacionCalor, ejemplo4, 10)
```




```
buenSuavizado(ecuacionCalor, ejemplo4, 10)
```

```
    'Buen suavizado'
```

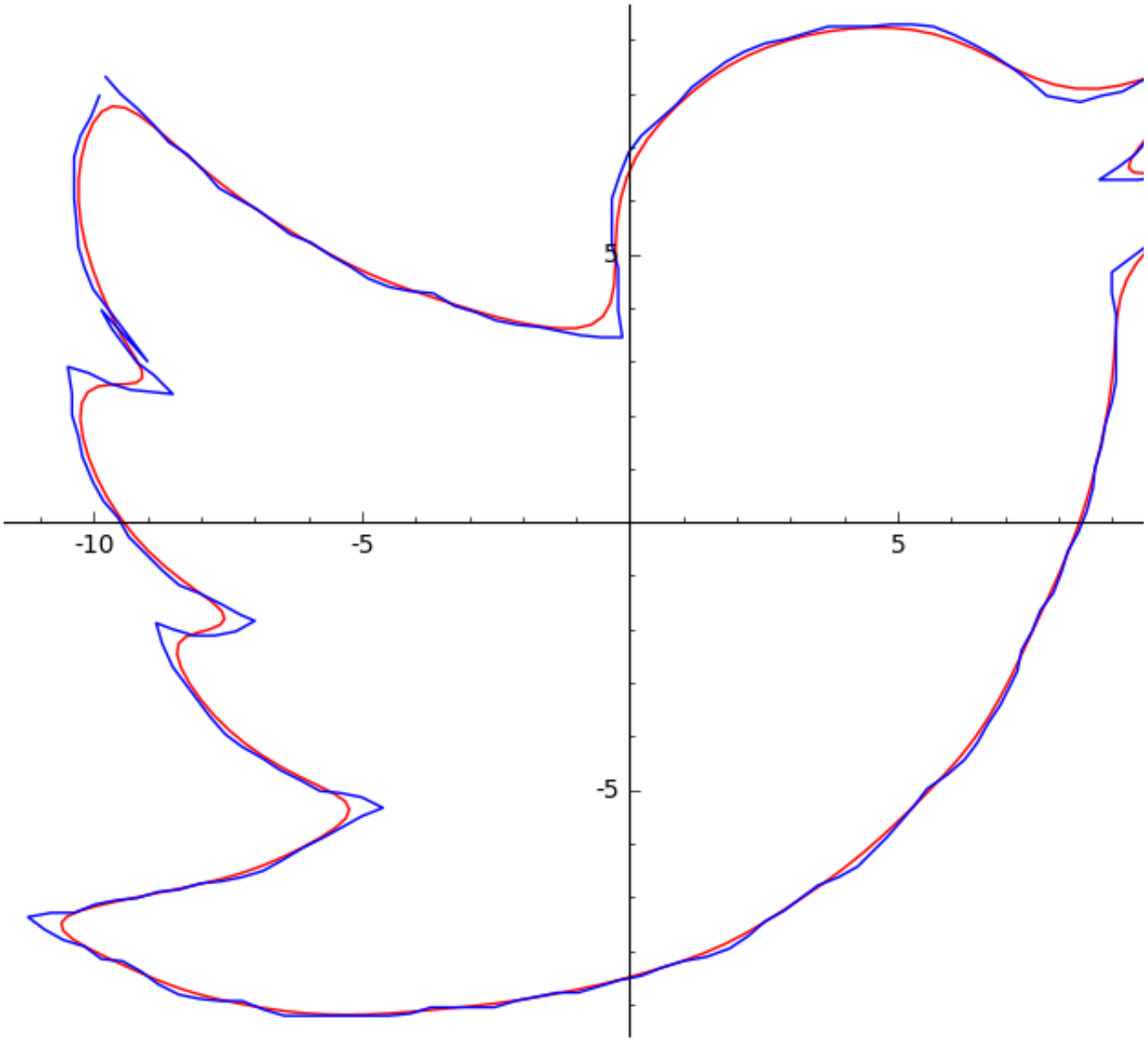
```
showPolygon(ecuacionCalor, ejemplo5, 50)
```




```
buenSuavizado(ecuacionCalor, ejemplo5, 50)
```

```
    'Buen suavizado'
```

```
showPolygon(ecuacionCalor, fixedEjemplo5, 20)
```




```
buenSuavizado(ecuacionCalor, fixedEjemplo5, 20)
```

```
'Buen suavizado'
```

En este último caso se ha aplicado el método a la curva poligonal sin las anomalías iniciales y el resultado es muy preciso.

METODO SUAVIZADO EXPONENCIAL

El método del suavizado exponencial aplica la siguiente formula sobre los puntos que generan la curva original:

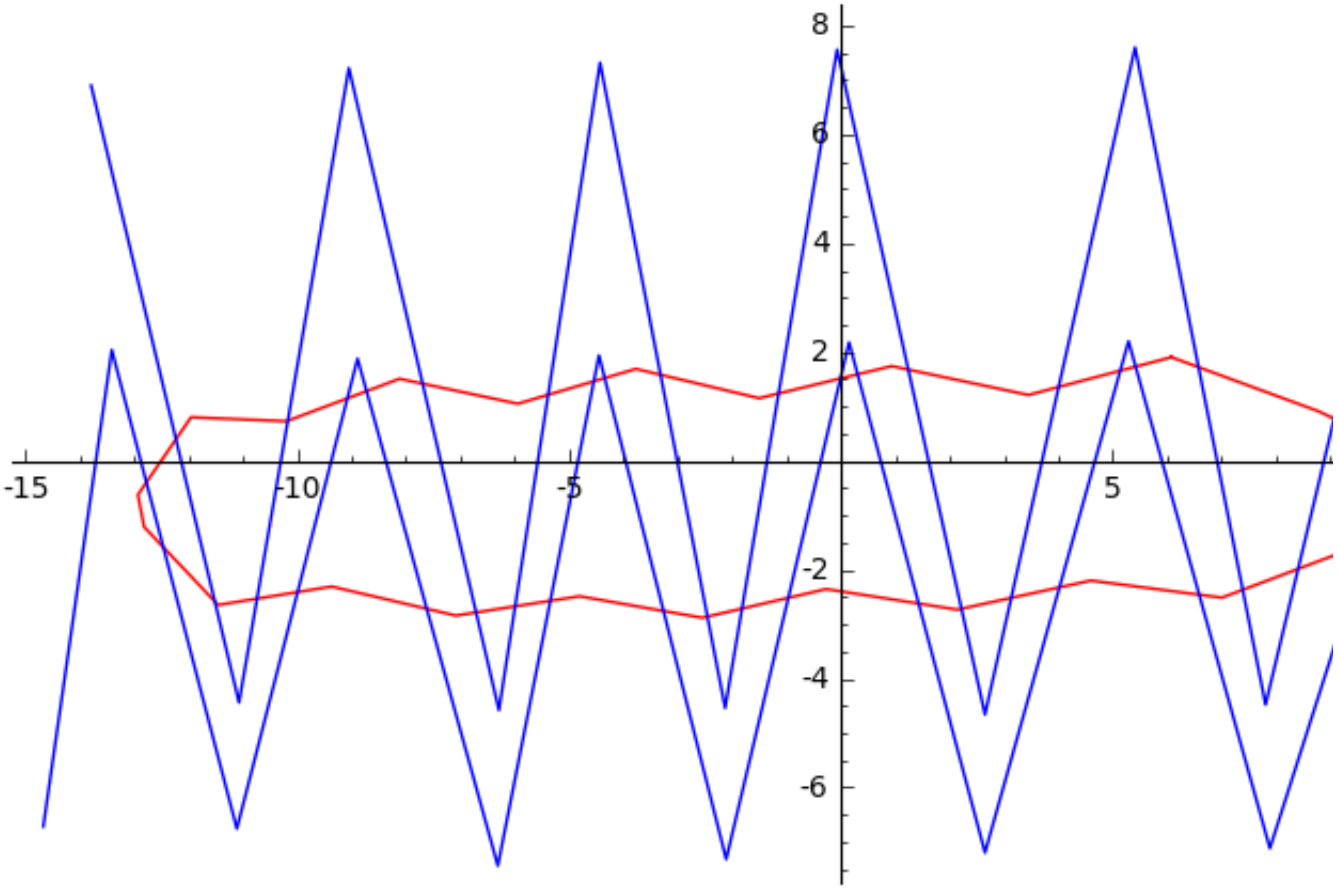
Esta fórmula esta creada para pronosticar la demanda de un producto en un periodo dado. Sin embargo, la hemos redefinido para que suavice las curvas poligonales a partir de los puntos que la forman.

```
def suavizadoExponencial(puntos):
    aux = calcularAlfa(puntos)
```

```
    alfa = 0.85
    primerox = puntos[0][0]+alfa*(puntos[1][0]-puntos[0][0])
    primeroy = puntos[0][1]+alfa*(puntos[1][1]-puntos[0][1])
    p = puntos[0]
    del puntos[0]
    puntos.append(p)
    lista=[[primerox,primeroy]]
    for i in range(1,len(puntos)):
        result1 = lista[i-1][0]+alfa*(puntos[i][0]-lista[i-1][0])
        result2 = lista[i-1][1]+alfa*(puntos[i][1]-lista[i-1][1])
        lista.append([result1,result2])
    return lista
```

Este método solo funciona para cuando alfa esta cerca a 1, si es más grande que uno no tiene ningún sentido y cuando es más pequeño que 0.5 tampoco.

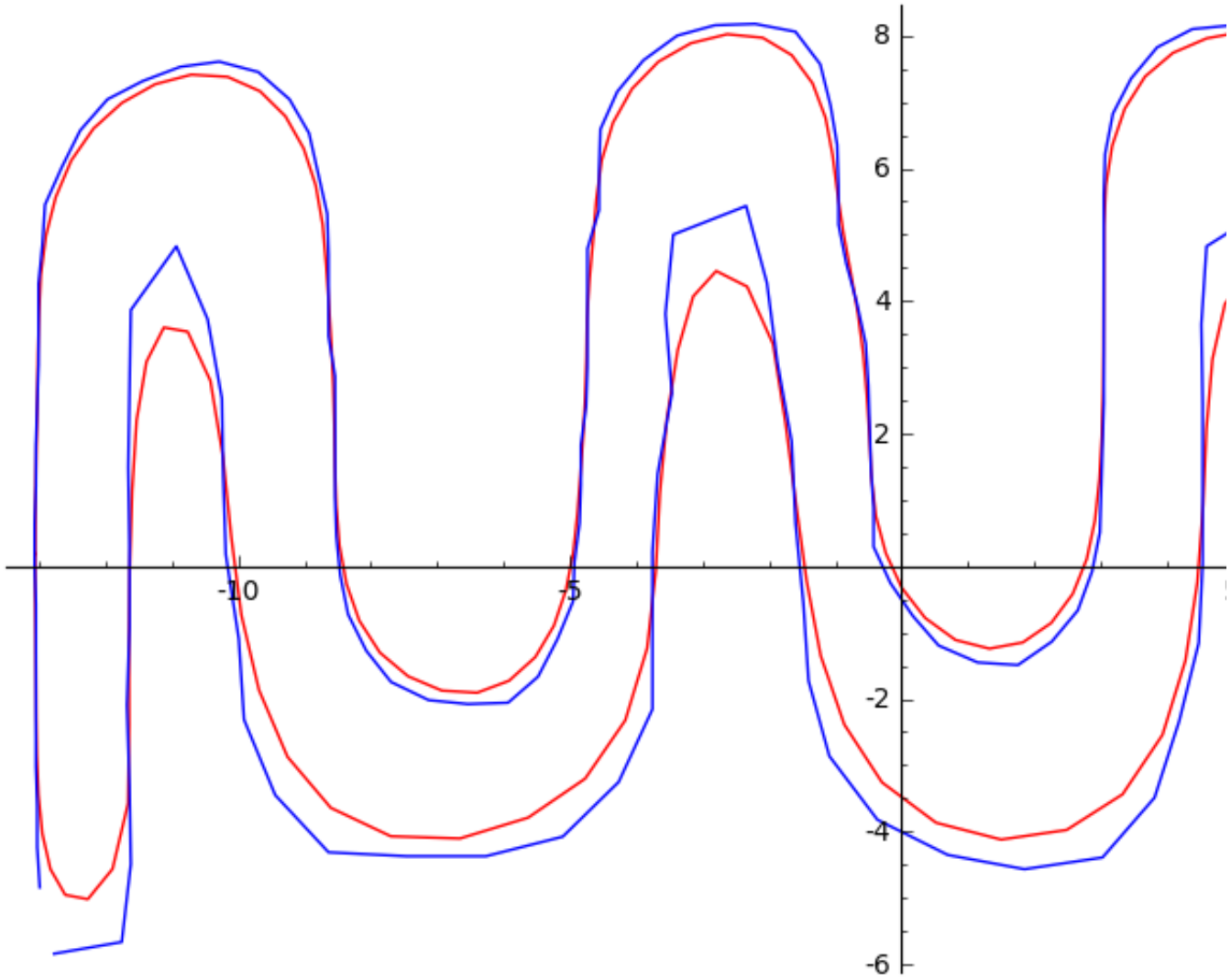
```
showPolygon(suavizadoExponencial, ejemplo1, 10)
```




```
buenSuavizado(suavizadoExponencial, ejemplo1, 10)
```

```
    'No buen suavizado'
```

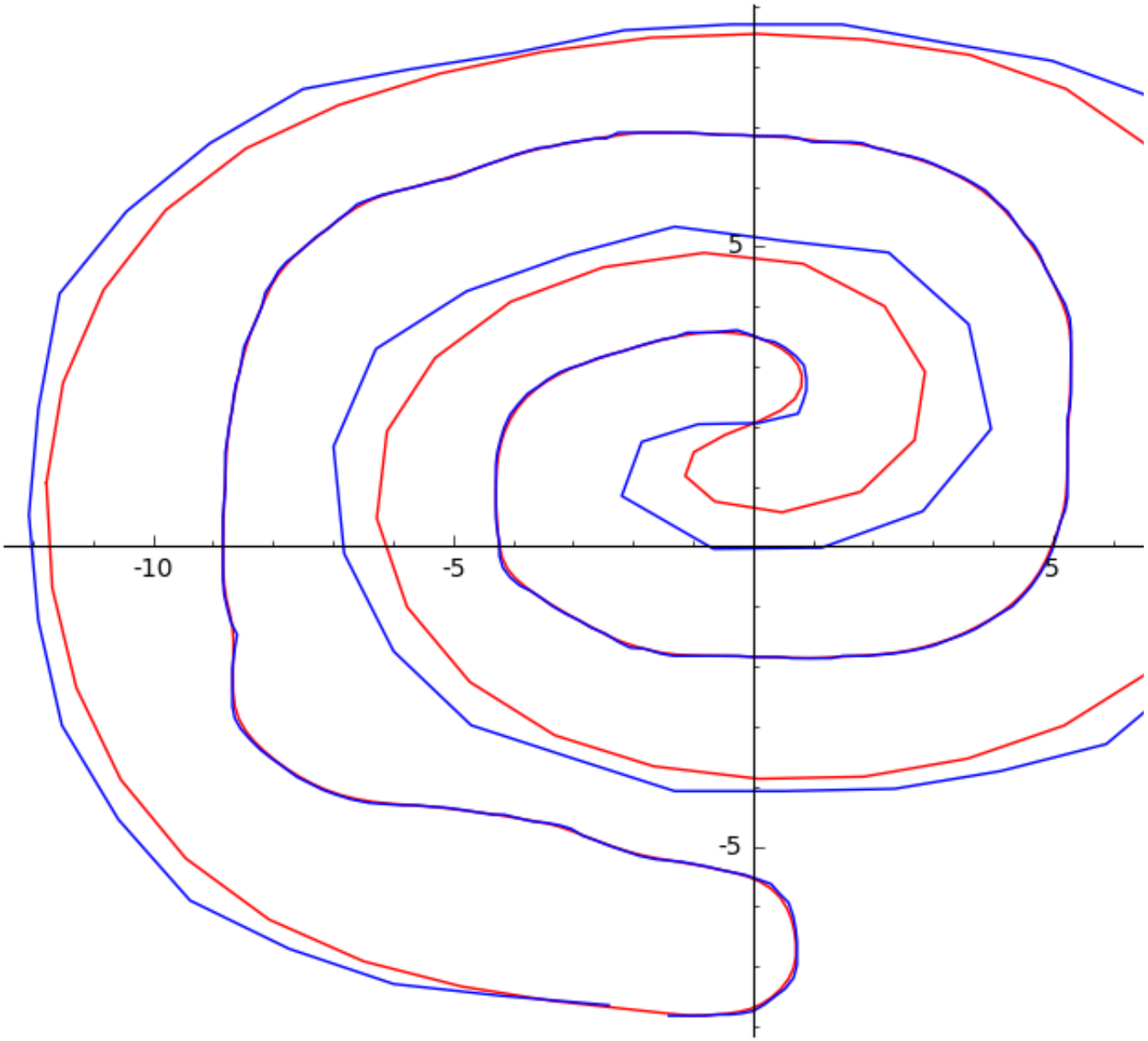
```
showPolygon(suavizadoExponencial, ejemplo2, 10)
```




```
buenSuavizado(suavizadoExponencial, ejemplo2, 10)
```

```
    'Buen suavizado'
```

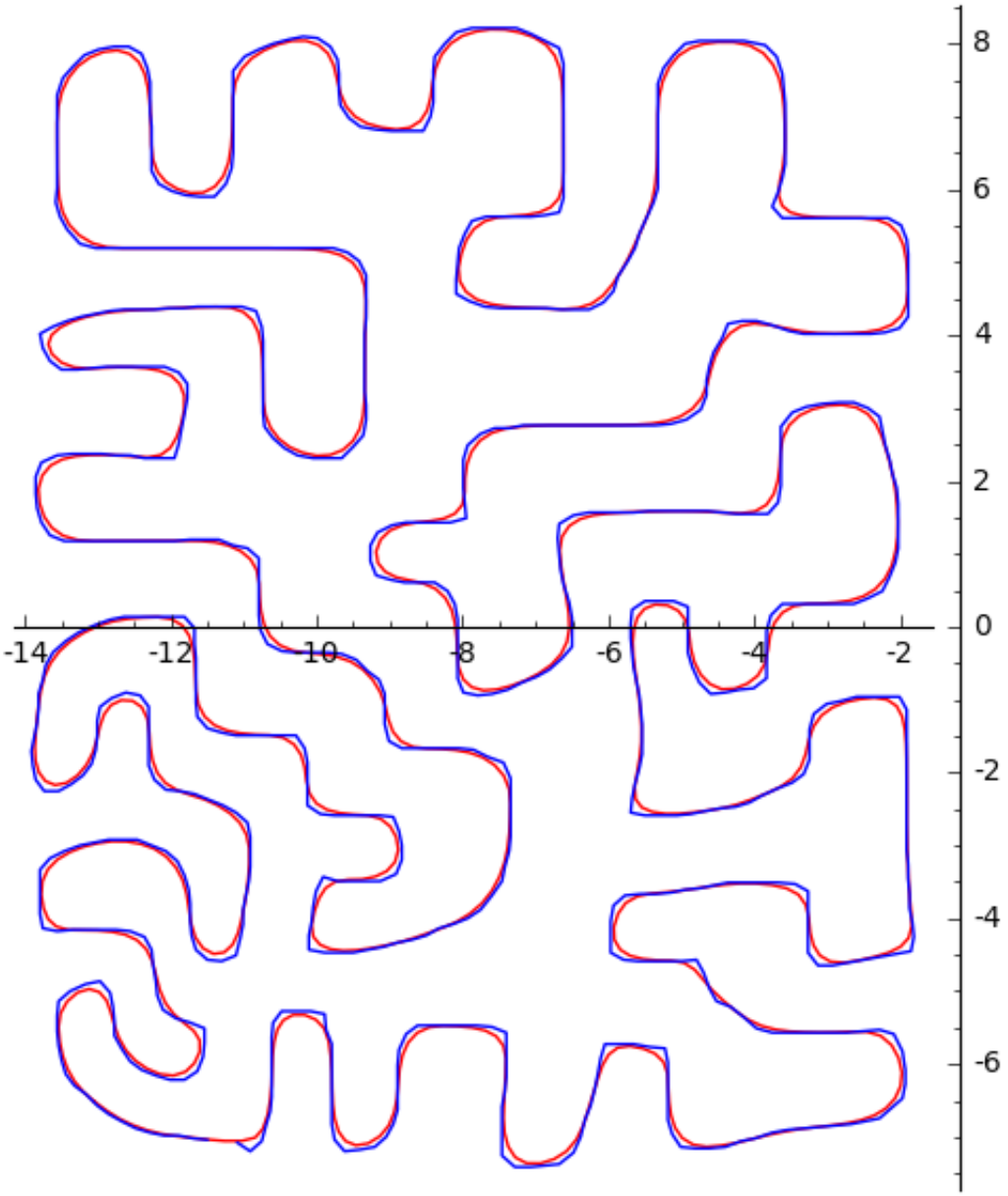
```
showPolygon(suavizadoExponencial, ejemplo3, 10)
```




```
buenSuavizado(suavizadoExponencial, ejemplo3, 10)
```

```
    'No buen suavizado'
```

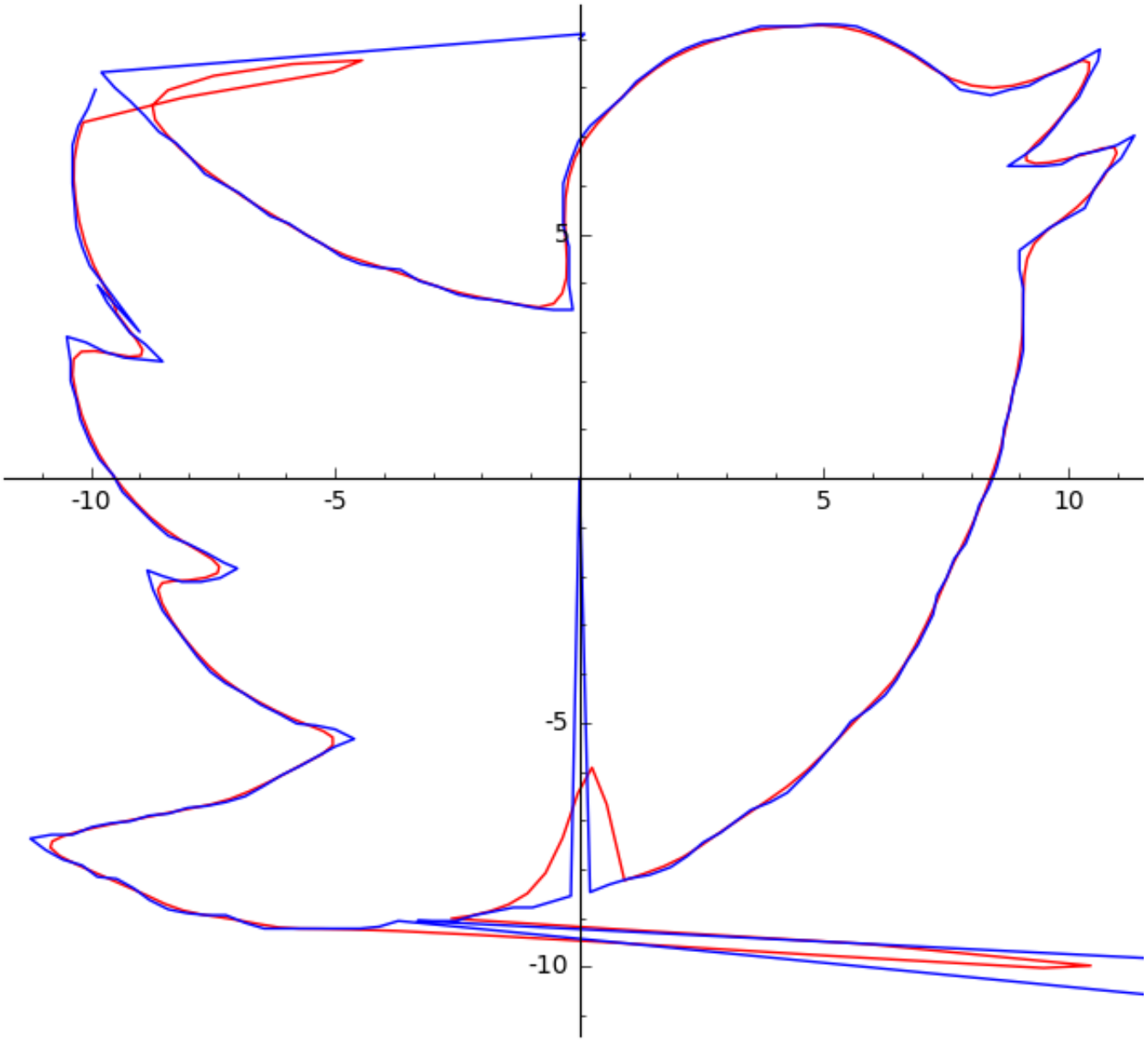
```
showPolygon(suavizadoExponencial, ejemplo4, 10)
```




```
buenSuavizado(suavizadoExponencial, ejemplo4, 10)
```

```
    'Buen suavizado'
```

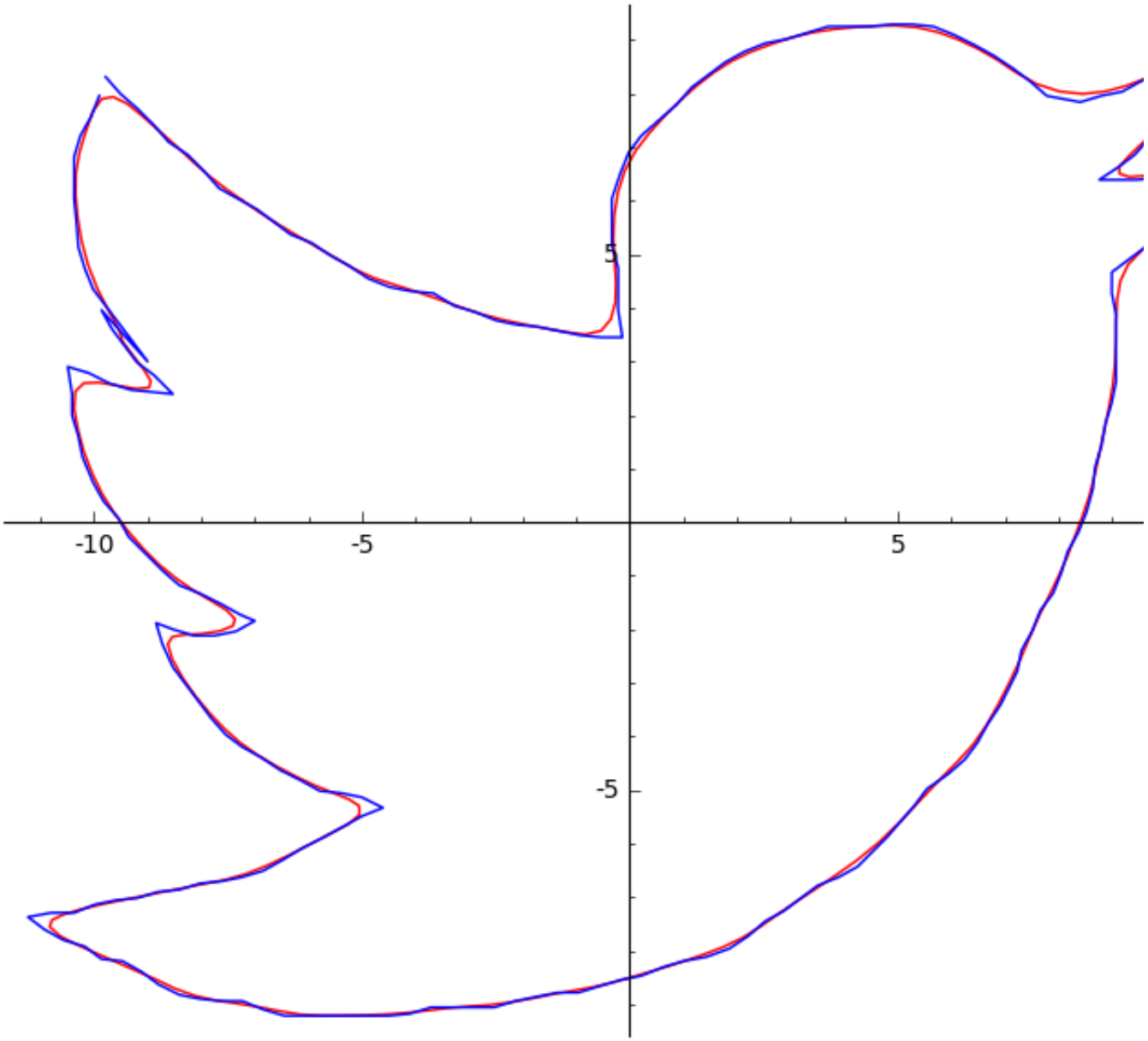
```
showPolygon(suavizadoExponencial, ejemplo5,10)
```




```
buenSuavizado(suavizadoExponencial, ejemplo5, 10)
```

```
    'Buen suavizado'
```

```
showPolygon(suavizadoExponencial, fixedEjemplo5,10)
```




```
buenSuavizado(suavizadoExponencial, fixedEjemplo5, 10)
```

```
    'Buen suavizado'
```

MÉTODOS DE GENERACIÓN DE CURVAS POLIGONALES ALEATORIAS

Este método genera poligonos aleatorios con los puntos desordenados.

```
import random
```

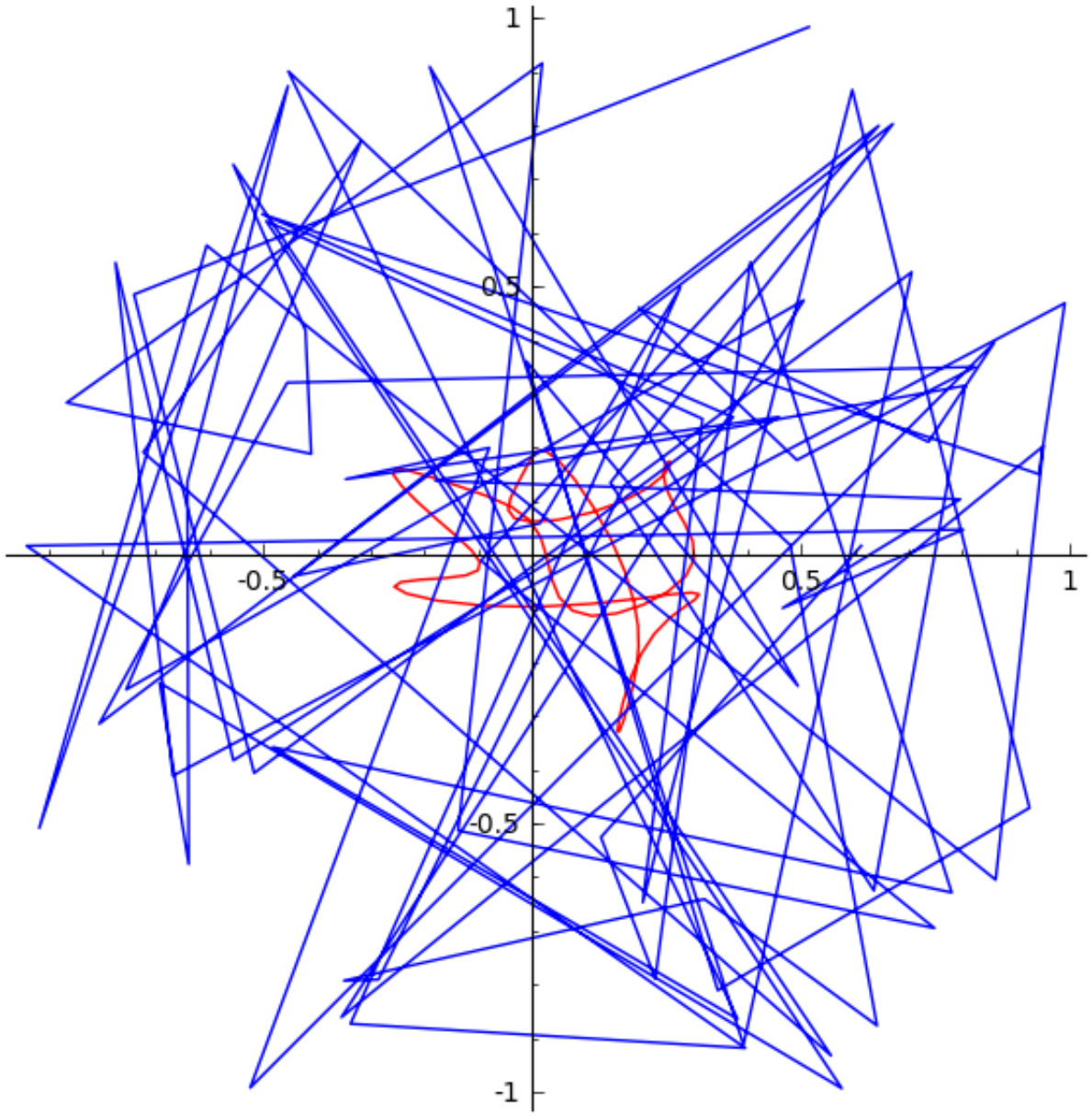
```
def polAleatorio():
    pol =[]
    numpuntos = random.randint(1,100)
    for i in range(numpuntos):
        pol.append([random.uniform(-1,1), random.uniform(-1,1)])
        i += 1
```

```
    return pol
```

Se muestra que esta curva poligonal es caótica, por tanto se necesita de un método que estructure esta curva poligonal de manera que se pueda analizar su suavidad.

Al mismo, además, le hemos aplicado el método de suavizado exponencial y vemos que no nos aporta información por la composición inicial que tiene.

```
showPolygon(suavizadoExponencial, polAleatorio(), 60)
```



Con este método se pueden ordenar cualquier lista de puntos y así visualizar una curva poligonal de forma estrellada, a la que se le puedan aplicar los suavizados implementados.

```
def angularSort(p, o):
    def compara(a,b):
        if a==b:
```

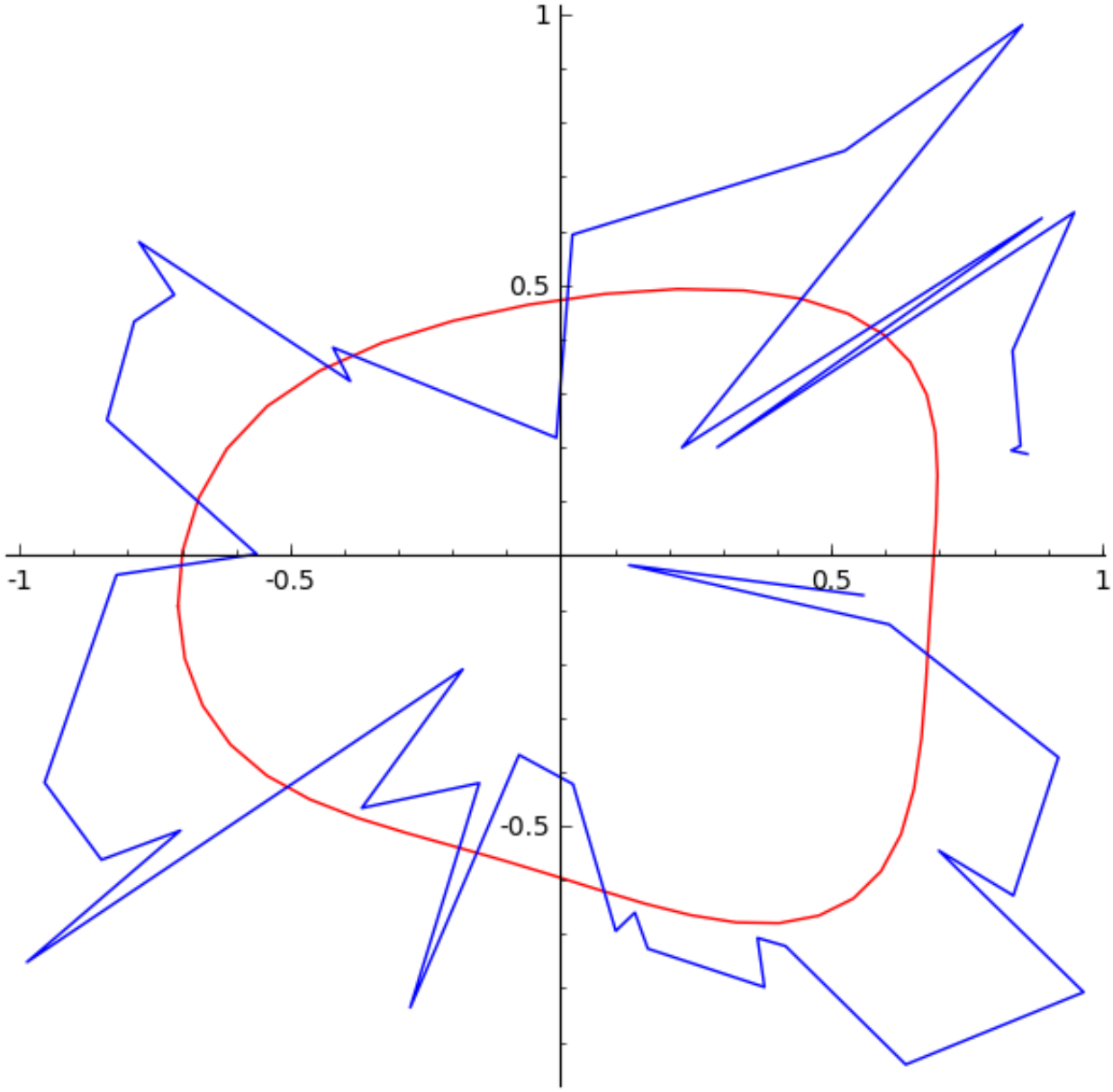
```
            return int(0)
        if a[1]>o[1] and b[1]<o[1]:
            return int(-1)
        if a[1]<o[1] and b[1]>o[1]:
            return int(1)
        if areaTriangulo(o,a,b)>0:
            return int(-1)
        if areaTriangulo(o,a,b)==0 and
distancia(a,o)>distancia(b,o):
            return int(-1)
        return int(1)

    po=sorted(p,cmp=compara)
    return po
```

```
def estrella(p,o):
    return angularSort(p,o)
```

```
EST = estrella(polAleatorio(), [0,0])
```

Aplicando el método de los puntos medios a la curva poligonal aleatoria con un número alto de iteraciones, se obtiene la curva suavizada.

```
showPolygon(puntosMedios, EST, 50)
```

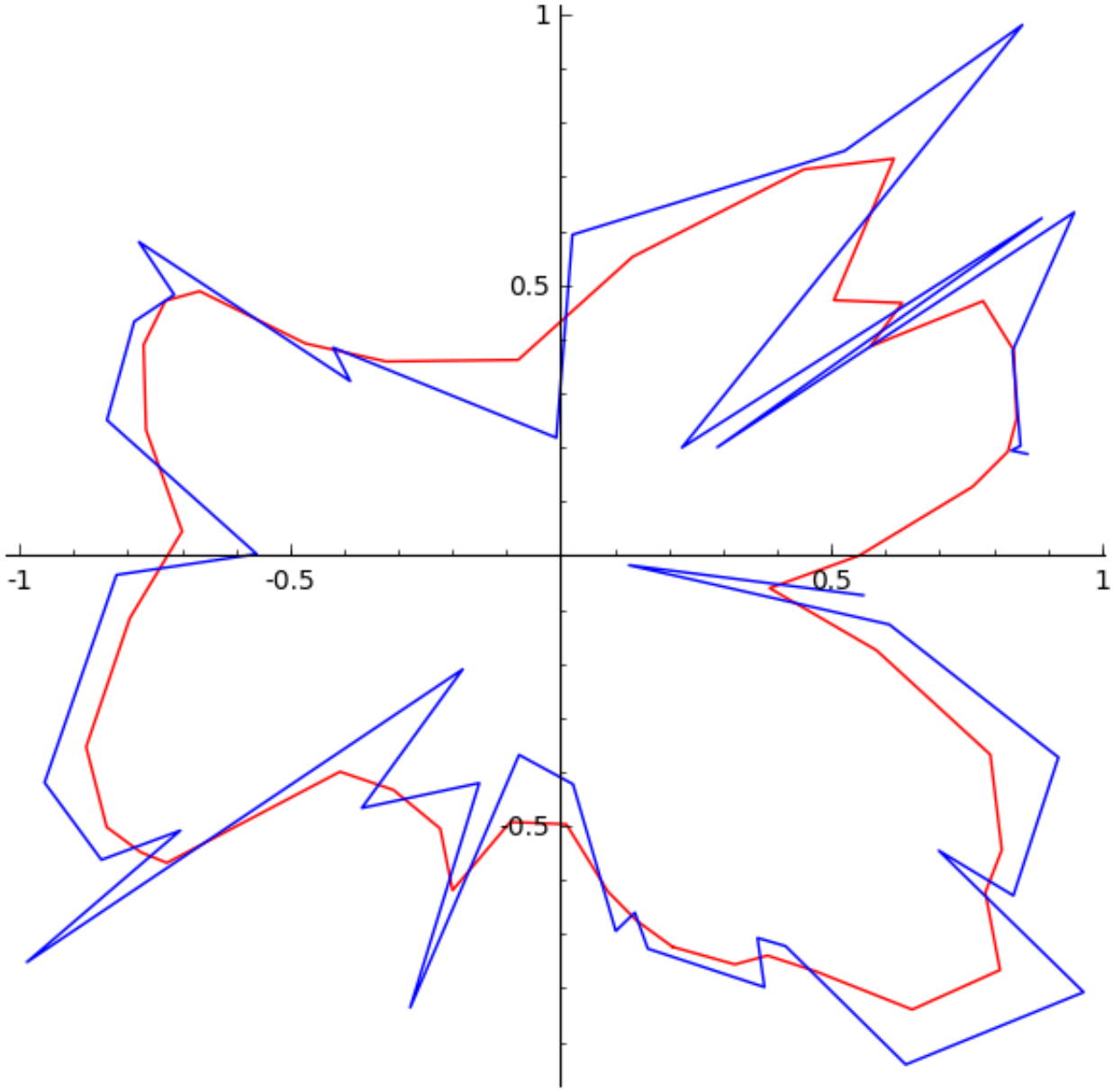



```
buenSuavizado(puntosMedios, EST, 50)
```

```
'Buen suavizado'
```

Ahora se prueba con el método de la ecuación del calor y se comporta de una manera más irregular el suavizado de la curva poligonal aleatoria.

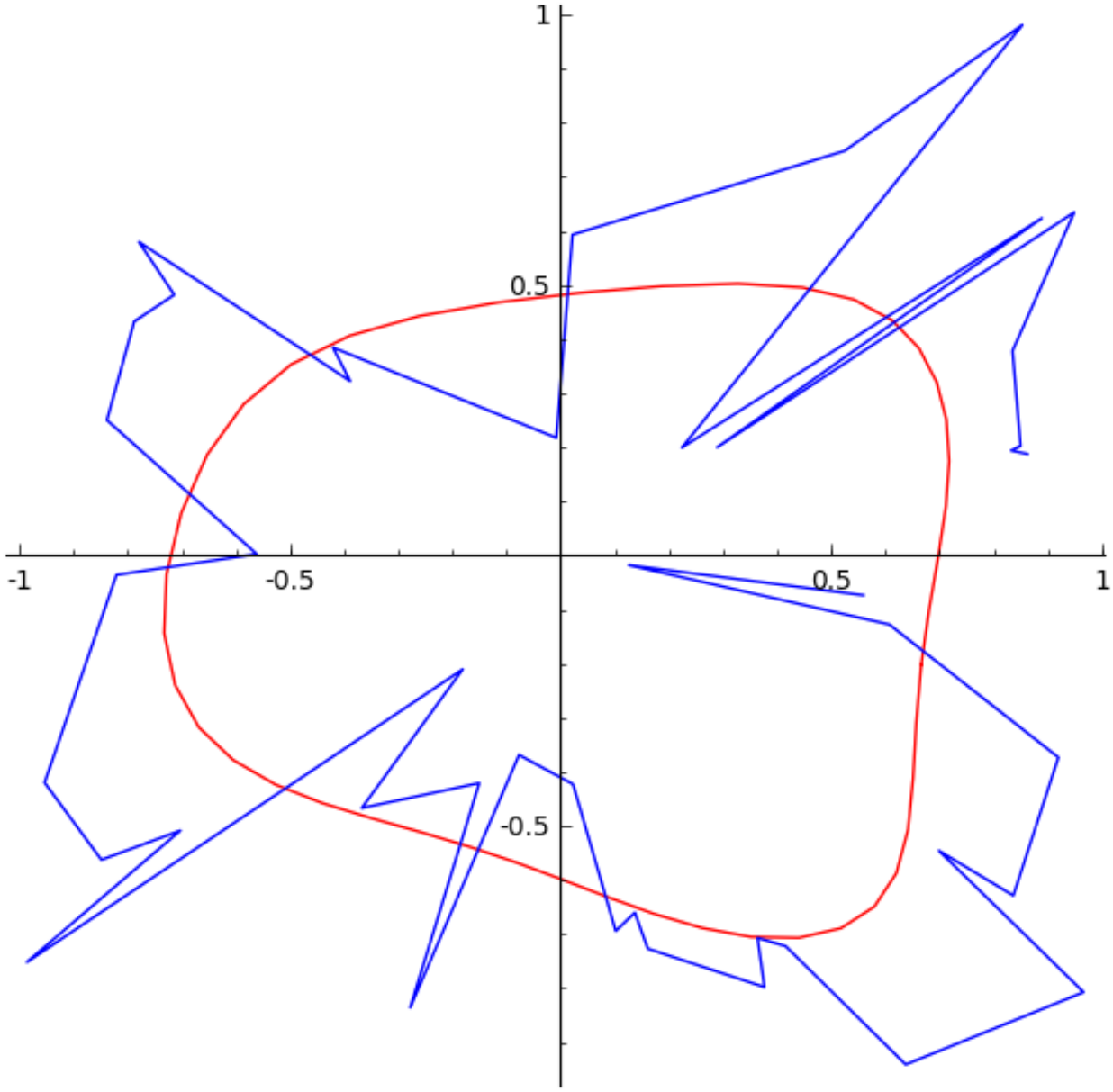
```
showPolygon(ecuacionCalor, EST, 50)
```



El último método que se aplicará será el de suavizado exponencial, que es el que mejor suaviza una curva poligonal como la que se está generando en forma de estrella.

Como los puntos se generan aleatoriamente, no se obtiene un suavizado tan visible como el que hemos visto en los ejemplos de cada método, pero este último es el que más se acerca a lo que queremos obtener.

```
showPolygon(suavizadoExponencial, EST, 50)
```



Como vemos los métodos que mejor ejecutan la técnica de suavizado son el suavizado exponencial y el método de los puntos medios. El método de la ecuación del calor provoca más fallos ya que su implementación es más compleja y no tan cercana a la ecuación original. A pesar de esto, en algunas figuras vemos que los tres métodos obtienen resultados muy buenos para el suavizado de la curva poligonal.

2. Bibliografía