Envolvente conexa: Algoritmo de fuerza bruta

De

Sergio Casado y Sandra Gómez

TABLA DE CONTENIDO

RAZONAMIENTO SEGUIDO EN EL ALGORITMO

Inicialmente comenzamos a buscar online formas para calcular envolventes conexas y los distintos algoritmos. Tras mucho investigar, para realizar un algoritmo de fuerza bruta llegamos a la conclusión de que debemos tratar cada posible segmento entre dos puntos cualesquiera de la nube de puntos como posible parte de la envolvente y comprobar si pertenece o no. La mejor forma de saber si un segmento que se encuentra entre dos puntos forma parte de la envolvente conexa, es ver si la recta que pasa por los dos puntos y divide el espacio en dos, deja a uno solo de los lados el resto de los puntos de la nube, es decir, si el resto de puntos están todos a la izquierda o a la derecha, ambos puntos pertenecen a la envolvente.

ALGORITMO

Teniendo ya claro como proceder con nuestro algoritmo, realizamos un esquema en seudocódigo del algoritmo pensado:

- Primero recorremos la nube de puntos eligiendo cada par de puntos posible
  - Comprobamos si los puntos pertenecen a la envolvente mirando si la recta que pasa por ambos puntos y divide el espacio deja el resto de los puntos, todos a la izquierda o todos a la derecha.
  - Si cumple esta condición ambos puntos pertenecen a la solución, es decir, a la envolvente conexa. Comprobando antes claramente si los puntos ya pertenecen o no a la envolvente.

Tras llegar a esta conclusión para nuestro código, se nos presentaba un problema, ¿Cómo sabemos si todos los puntos están a un lado o al otro de una recta? Para solucionarlo, dimos con un método usando la pendiente de las rectas y que explicamos a continuación.

METODO POR PENDIENTES

Para llevar a cabo este método, lo primero a verificar el signo que tiene la pendiente de la recta que pasa por los dos puntos preseleccionados. Tras esto, elegimos uno de los dos puntos, en este caso elegimos el que se encuentra mas abajo, en caso de estar a la misma altura, elegimos el que se encuentra mas a la izquierda. Utilizando este punto, formamos rectas que pasan por cada uno del resto de puntos y calculamos sus pendientes. Teniendo la pendiente calculada originalmente y la nueva pendiente fruto del nuevo punto, podemos saber si ese punto se encuentra a la izquierda o a la derecha de la recta siguiendo la regla explicada abajo.

REGLA DEL SIGNO Y ORDEN DE PENDIENTE

Primero vamos a separar en dos casos muy diferentes, el primero es cuando la pendiente original es menor que cero:

- Después, separamos los casos en función de si se encuentran a la izquierda o a la derecha del punto inferior elegido:
  - Si el nuevo punto se encuentra a la izquierda del punto inferior: si la nueva pendiente es mayor que la original, el nuevo punto se encuentra a la izquierda de la recta, si la pendiente es menor que la original se encuentra a la derecha.
  - Si el nuevo punto se encuentra a la derecha del punto inferior: si la nueva pendiente es mayor que la original, el nuevo punto se encuentra a la derecha de la recta, si la pendiente es menor que la original se encuentra a la izquierda.

Veamos el caso en que la pendiente original es mayor o igual a cero:

- Volvemos a separar en función de si se encuentran a la izquierda o a la derecha del punto inferior elegido:
  - Si el nuevo punto se encuentra a la derecha del punto inferior: si la nueva pendiente es mayor que la original, el nuevo punto se encuentra a la izquierda de la recta, si la pendiente es menor que la original se encuentra a la derecha.
  - Si el nuevo punto se encuentra a la izquierda del punto inferior: si la nueva pendiente es mayor que la original, el nuevo punto se encuentra a la derecha de la recta, si la pendiente es menor que la original se encuentra a la izquierda.

Con esto, ya tenemos como saber si los puntos se encuentran a un lado o al otro de una recta y por tanto podemos usarlo en nuestro algoritmo para buscar la envolvente conexa. Veamos el código:

CODIGO EN R

```
algFuerzaBruta<-function(p){
  solucion<-array()
  for (i in 1:(length(p)/2)) {
    for(j in 1:(length(p)/2)){
      if(i!=j){
        #true significa que no hay puntos a la izq o der, false que hay al menos un punto
        izq=TRUE
        der=TRUE
        m<-pendiente(p[i,1:2], p[j,1:2])
        #pequeña derecha
        #grande izq
        for(k in 1:(length(p)/2)){
          if((k!=i)&&(k!=j)){
            x<-bajo(p[i,1:2], p[j,1:2])
            m1<-pendiente(p[k,1:2], x)
            if(m<0){
              if(comprobar(x,p[k,1:2])){ #true derecha y false izq
                if(m1>m){
                  der=FALSE
                }else if(m1<m){
                  izq=FALSE
                }
              }else{
                if(m1<m){
                  der=FALSE
                }else if(m1>m){
                  izq=FALSE
                }
              }
            }else{
              if(comprobar(x,p[k,1:2])){ #true derecha y false izq
                if(m1<m){
                  der=FALSE
                }else if(m1>m){
                  izq=FALSE
                }
              }else{
                if(m1>m){
                  der=FALSE
                }else if(m1<m){
                  izq=FALSE
                }
              }
            }
          }
        }
        if(izq||der){
          if(length(solucion)==1){
            solucion<-array(c(p[i,1],p[j,1],p[i,2],p[j,2]),dim = c(2,2))
          }else{
            if(!pertenece(solucion, p[i,1:2])){
              solucion<-array(c(solucion[1:(length(solucion)/2),1],p[i,1],
            }
            if(!pertenece(solucion, p[j,1:2])){
              solucion<-array(c(solucion[1:(length(solucion)/2),1],p[j,1],
            }
          }
        }
      }
    }
  }
  return(solucion)
}
```

```
solucion[1:(length(solucion)/2),2],p[i,2]),dim = c((length(solucion)/2)+1,2))

solucion[1:(length(solucion)/2),2],p[j,2]),dim = c((length(solucion)/2)+1,2))
```

Estas primeras fotos son el codigo principal del algoritmo, hacemos uso de funciones auxiliares para no llenar el algoritmo de codigo dificil de leer y poder facilitar su comprension.

```
pendiente<-function(p,q){
  if(p[1]>q[1]){
    m<-((p[2]-q[2])/(p[1]-q[1]))
  }else{
    m<-((q[2]-p[2])/(q[1]-p[1]))
  }
  return(m)
}
bajo<-function(p,q){ #Cogemos el pur
  if(p[2]<q[2]){
    return(p)
  }else if(p[2]>q[2]){
    return(q)
  }else if(p[1]<q[1]){
    return(p)
  }else{
    return(q)
  }
}
```

Con estas dos funciones calculamos la pendiente de la recta dados dos puntos, y calculamos cual es el punto mas bajo dado dos puntos.

```
comprobar<-function(x,p){ #true derecha y false izq
  if(x[1]<p[1]){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
pertenece<-function(solucion, p){
  per=FALSE
  for(i in 1:(length(solucion)/2)){
    if((p[1]==solucion[i,1])&&(p[2]==solucion[i,2])){
      per=TRUE
    }
  }
  return(per)
}
```

Con estas ultimas funciones, calculamos si un punto se encuentra a la derecha o a la izquierda de otro punto, y comprobamos si un punto pertence a un array denominado solucion.

Para realizar una introduccion de los datos en nuestro algoritmo, se debe realizar de la siguiente manera, para evitar problemas en su ejecucion:

```
X<-c(-2,0,4,-1,3,6,2,5,-3,7)
Y<-c(2,2,4,1,-5,5,5,-8,2,3)
p<-array(c(X,Y),dim = c(10,2))
algFuerzaBruta(p)
```

Siendo X e Y, los valores de las coordenadas de cada punto y p la nube de puntos.