Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Modelación y Computación Gráfica para Ingenieros

Tarea 1

Radiación térmica de planta industrial

Autor: Javier Gómez P.
Profesor: Nancy Hitschfeld K.
Auxiliares: Pablo Pizarro R.
Pablo Polanco
Ayudantes: Joaquín T. Paris
Roberto I. Valdebenito
Rodrigo Ramos T.

Fecha de realización: 16 de abril de 2017
Fecha de entrega: 16 de abril de 2017
Santiago, Chile

Índice de Contenidos

1. Introducción

A lo largo de la historia, los computadores han ido avanzando en su capacidad y tecnología. Gracias a estos avances, los computadores actuales pueden procesar una gran cantidad de tareas en cortos tiempos. Con esto, los matemáticos han podido utilizar las herramientas que ofrecen estos para resolver y modelar problemas de la vida real que requieren de una gran cantidad operaciones que, sin un computador, tardarían mucho en calcularse.

En este trabajo se simulará una planta industrial la cual emitirá calor a la atmósfera. Se modelará el comportamiento térmico de esta, para así poder ver el impacto ambiental que genera.

Para modelar el comportamiento térmico, se usará una matriz que representará un perfil de 4 km de ancho y 2 km de alto de alguna porción del litoral del país. Por lo tanto, esta matriz tendrá tamaño $4000 \ / \ h \times 2000 \ / \ h$ elementos, con h el tamaño de cada punto de la matriz (es decir, cuantos metros representará cada uno de los elementos de la matriz).

Los mayores problemas a resolver son: la generación del terreno y el cálculo de la temperatura para cada punto del perfil.

2. Diseño de la solución

Para modelar el problema introducido anteriormente, se utilizó una discretización de $h = 1\ m$, ya que esta permitía que la simulación tardara un tiempo adecuado (alrededor de 5 minutos) y también lograba una definición aceptable del problema (la cual se revisará más adelante).

Además de utilizar una descritización adecuada, el problema requería de la generación aleatoria de un terreno, solución numérica para las temperaturas en cada instante de tiempo, y programación para graficar dicha solución en el terreno generado.

Para lograr implementar esta simulación, fue necesaria la creación de una clase *Litoral*, cuya implementación se puede revisar en el anexo 1.

2.1. Generación del terreno

Para simular un perfil adecuado de una porción del litoral, se generó un terreno aleatorio, que contuviera entre un 30 % y un 40 % de agua, seguido de una cadena de cordilleras generadas aleatoriamente en forma y altura.

1. Primero, se creó un objeto de la clase *Litoral*, el cual contendría seis matrices (una por cada tiempo requerido), la discretización elegida (h), las variables de tiempo y el número de iteraciones a realizar cuando se calcule la solución numérica.

2. Se calculó un porcentaje entre 30 % y 40 % que sería la cantidad de agua que contendría el perfil. Luego se calculó el limite (en coordenadas de la matriz) que separaría el agua del continente, y se guardó en la variable *limite*.

3. Se rellenó el 10 % de la altura de la matriz (y entre el 30 % y 40 % de ancho, según la variable *limite*) con la condición de borde (T^{o} del agua), utilizando una función $tagua(t)$, definida en la misma clase.

Al graficar la matriz en este punto, se tendría un rectángulo de $(4000\ /\ h) \times (2000\ /\ h)$ elementos. Es importante destacar que si la matriz es muy grande, los cálculos que se realizarán posteriormente tomarán una gran cantidad de tiempo.

Para generar las cordilleras:

1. Se creó un arreglo del tamaño del ancho de la cadena de montañas (que corresponde a la diferencia del ancho de la matriz y la variable *limite*, dejando espacio libre para luego posicionar la planta térmica). Cada elemento de este arreglo corresponde a la altura de la cadena de montañas en cada columna de la matriz donde no haya agua.

2. Se fijó el valor de todas sus variables igual a la mitad del alto de la matriz, creando así un bloque de montañas.

3. Si el parámetro *alpha* del objeto de clase *Litoral* es igual a 1, se generarán cordilleras. Si es igual a 0, el bloque continental será plano, y de esa manera, se salta hasta el item n^{o} 7.

4. Se subdividió el arreglo en diez partes iguales, guardando los índices de los elementos que separaban cada una de las subdivisiones.

5. Se modificó el valor de dichos índices aleatoriamente, aumentando (o disminuyendo) entre el 0 % y el 100 % de su valor.

6. Usando la función denominada *pendiente* (implementada en el mismo módulo, fuera de la clase), se rellenó el resto del arreglo de manera que los valores formaran una especie de 'regresión lineal'.

7. Se rellenó la matriz con los datos que indicaba el arreglo. Ya que cada elemento del arreglo representaba una columna de la matriz donde había cordillera, dicha columna se llenó con NaN desde el borde inferior de la matriz, hasta la altura que indicara el valor en el arreglo.

Una vez que ya hemos generado la cordillera, debemos generar la planta, la cual ubicaremos justo sobre los primeros (100 / h) elementos despues de la coordenada *limite* en nuestra matriz. Esto debido a que la planta mide 100 [m] de ancho, y según nuestra discretización h, debe ocupar dicha cantidad de elementos.

2.2. Solución Numérica

Se consideró que la distribuición de temperatura en la atmósfera cumple con la ecuación de Laplace. Siendo $T(x, y)$ la temperatura en un punto (x, y) se cumple que:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{2.1}$$

Luego, utilizando los datos del servicio marítimo, se determinó que la temperatura del mar es variable a lo largo del tiempo, pero no es influenciable por cambios de temperatura a nivel atmosférico.

A continuación se observa el comportamiento del agua en función del tiempo.

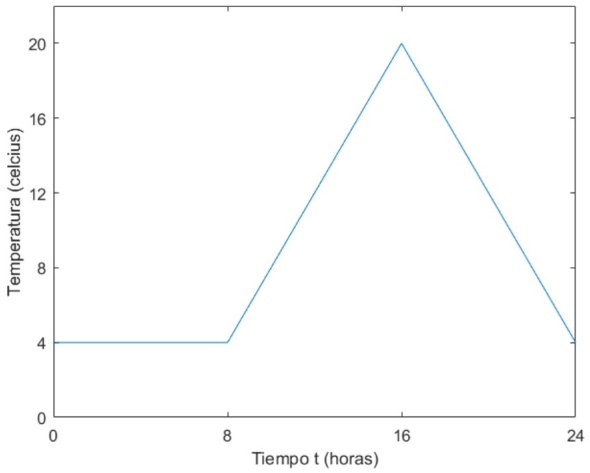

Figura 2.1: Temperatura del agua a lo largo del tiempo

También se sabe que la temperatura de la atmósfera es variable, y en condiciones normales, posee igual variación que la temperatura del agua, pero esta decae linealmente 6ºC cada 1000 [m]. Por lo tanto, la temperatura inicial de la atmósfera dependerá del tiempo que estemos utilizando.

Para la planta térmica, el calor emitido posee el siguiente comportamiento:

$$T = 500 \cdot (\cos(\frac{t \cdot \pi}{12}) + 2)[^oC] \tag{2.2}$$

Al calcular la temperatura de los puntos, se considerará la temperatura de las cordilleras igual a 15 [ºC].

Ya que la temperatura del agua no varía en un tiempo fijo, esta es una condición de borde de tipo Dirichlet, por lo que el problema de calcular la temperatura de la atmósfera no tiene condiciones de borde de Neumann, así se puede resolver de manera iterativa.

$$U_{i,j} = \frac{1}{4} \cdot (U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1}) \tag{2.3}$$

Para cada punto en la matriz, su valor dependerá de los puntos aledaños al mismo. En el caso de los bordes, su valor no se ve afectado por el calor emitido por la planta, por lo que no es necesario calcular su valor promediando los puntos aledaños.

Si bien se tiene la fórmula para calcular el valor de cada punto, este no es el valor definitivo ya que las diferencias de temperaturas entre la planta, el agua y la atmósfera es muy grande. Para poder llegar a un valor 'cercano' al real, se debe realizar este cálculo numerosas veces. A este número se le llamará *'número de iteraciones'*, y corresponderá a un parámetro de la clase *Litoral*. Se usará un número de iteraciones igual a 100, esto ya que al calcular 100 veces la temperatura de cada punto, se puede acercar al resultado 'final'.

3. Resultados

Para el modelo, se utilizaron distintos tiempos (que representan la hora del día). Los valores utilizados fueron $t = 0$, $t = 8$, $t = 12$, $t = 16$, $t = 20$.

A continuación se muestran los gráficos para el caso base (sin la planta térmica) y para cada t distinto.

1. Caso Base

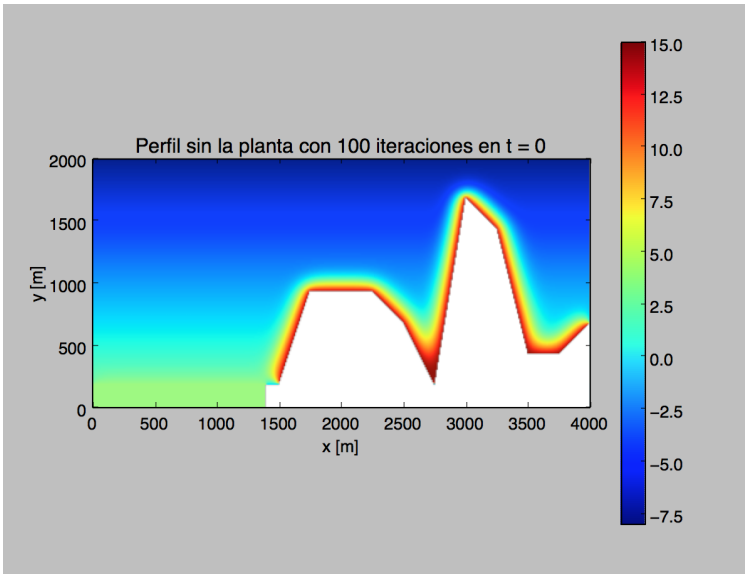

Figura 3.1: Modelo sin la planta térmica, en t = 0, con 100 iteraciones y $alpha = 1$

Cabe destacar que la porción de tierra plana que se encuentra justo antes de la coordenada 1500 [x] corresponde a la ubicación de la planta térmica.

Se puede apreciar cómo la temperatura de la cordillera afecta a la temperatura de la atmósfera, al igual que la geometría de la cadena de montañas.

2. Caso t = 0

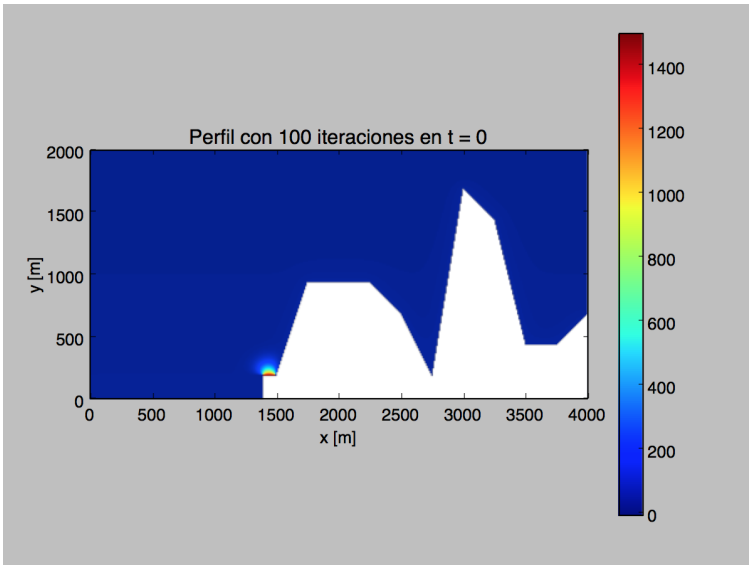

Figura 3.2: Modelo con temperatura emitida de la planta térmica en t = 0, 100 iteraciones y $alpha = 1$

3. Caso t = 8

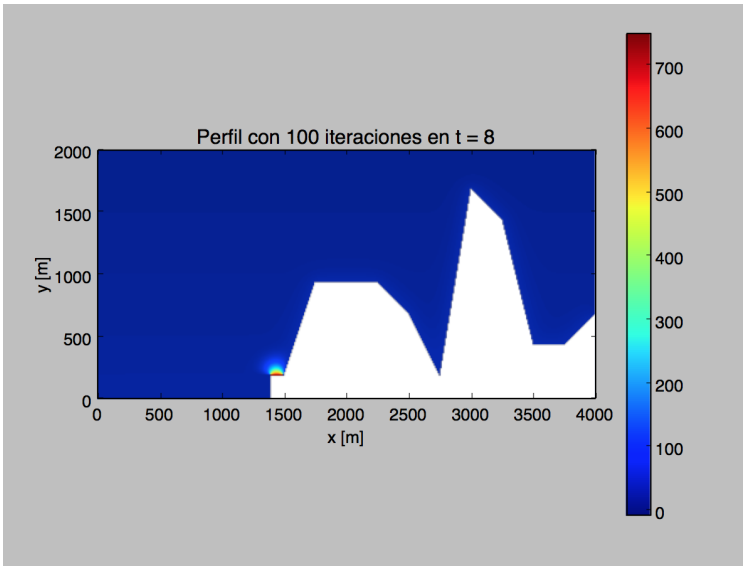

Figura 3.3: Modelo con temperatura emitida de la planta térmica en t = 8, 100 iteraciones y $alpha = 1$

4. Caso t = 12

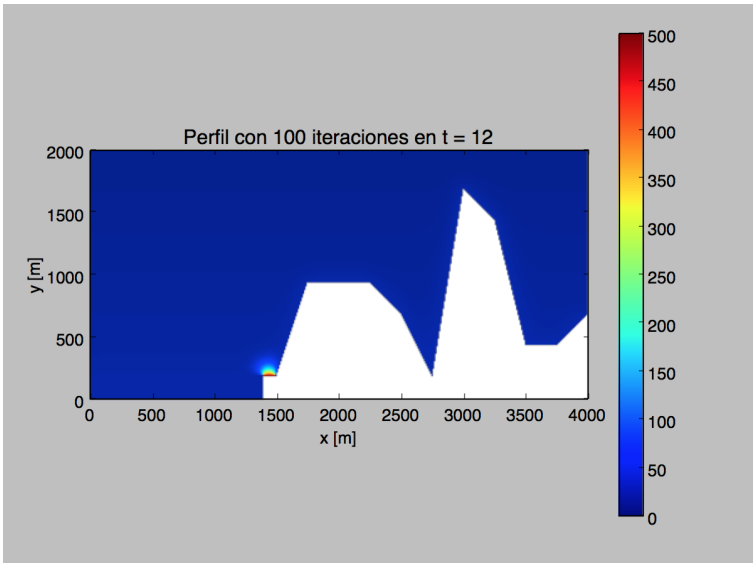

Figura 3.4: Modelo con temperatura emitida de la planta térmica en t = 12, 100 iteraciones y $alpha = 1$

5. Caso t = 16

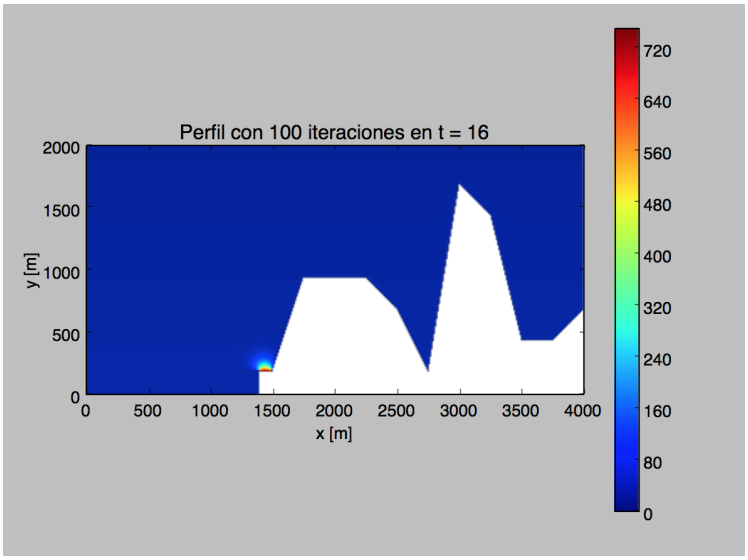

Figura 3.5: Modelo con temperatura emitida de la planta térmica en t = 16, 100 iteraciones y $alpha = 1$

6. Caso t = 20

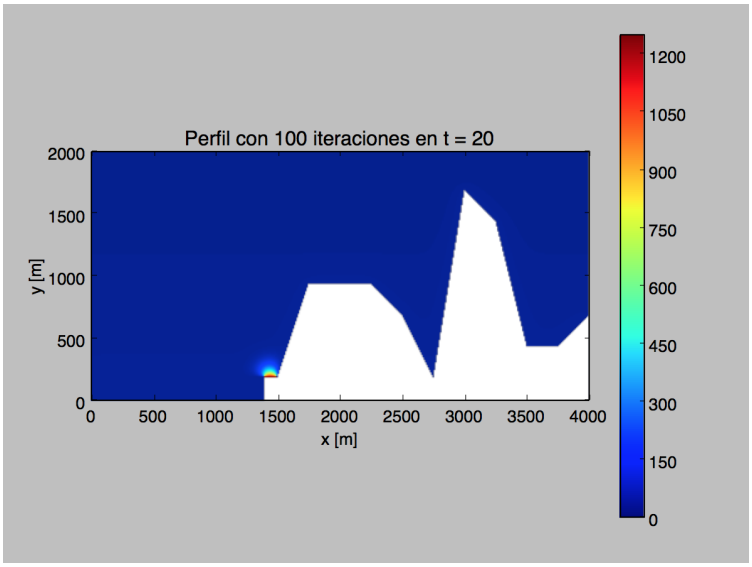

Figura 3.6: Modelo con temperatura emitida de la planta térmica en t = 20, 100 iteraciones y $alpha = 1$

Al analizar los resultados, se puede ver que son consistentes. Los puntos lejanos a la fuente de calor se mantienen en una 'baja' temperatura (comparada con la fuente de calor), por lo que gran parte del paisaje se ve del mismo color (azul oscuro), debido a la escala de colores en función de la temperatura (ver figuras), fenómeno que no pasa en el caso base, que no incluye la planta térmica.

También se puede observar un pequeño 'arcoiris' al rededor de la planta térmica. Esto se produce por la disicpación del calor, y refleja como decae la temperatura en las cercanías de la planta.

En cada tiempo distinto, la temperatura del mar varía, por lo que en cada figura la temperatura del mar es distinta. Pero como se puede observar, esto no afecta de forma severa al sistema, ya que la diferencia de temperatura es muy grande. Si bien la temperatura de la atmósfera depende de la temperatura del mar, al graficar el sistema, no se puede notar ya que la planta concentra gran parte del calor.

Al enfocarse en la geometría de las cordilleras, a simple vista en los gráficos no se ve que afecte a la solución, pero si se considera el método usado para realizar el cálculo de la temperatura en la atmósfera, si debiera afectar, ya que la cordillera tiene una temepratura fija (15 ºC). Esto representaría una condición de borde en el problema, por lo tanto, afectaría la forma y altura de la cordillera, pero que por la diferencia de temperatura, no es observable en el gráfico.

Si se comparan los gráficos entre si, se puede notar que el sistema con menor temperatura media es el que se encuentra en t = 12 (Figura 3.4). Esto es lógico, ya que en t = 12 la temperatura emitida por la planta es considerablemente menor que en los otros casos. La temperatura del agua en este caso es más alta que en otros casos, pero la diferencia es muy pequeña.

4. Conclusiones

Del problema anterior y su solución, se puede concluir que:

1. Es importante escoger una discretización h adecuada, ya que de esta dependerá el tamaño de las matrices que se generarán y operarán, y también el tiempo que tomarán dichas operaciones. En esta ocasión, la simulación tomó alrededor de 5 minutos, pero con una discretización más aguda (h más pequeño) tomaría mucho más tiempo.

   Si bien una discretización más fina otorga más detalle, para este problema no era necesario, ya que como se puede observar en la sección de resultados, los gráficos obtenidos son muy similares por lo que un poco más de detalle no otorgaría más información relevante, pero si aumentaría el tiempo de ejecución.

2. Para resolver problemas con condiciones de borde tipo Dirichlet, se puede realizar una aproximación utilizando un método iterativo para la ecuación de Laplace. También se puede resolver utilizando el método de Gauss, pero este aumenta considerablemente la implementación del método. Este último es conveniente ocuparlo sólo cuando el problema tenga condiciones de borde de tipo Neumann.

3. En el problema al analizar los gráficos se puede ver que, la cordillera al afectar la temperatura de la atmósfera, sirve para amortiguar en menor medida los efectos de la planta térmica. Sin embargo, la temperatura emitida por la planta térmica es muy alta, por lo que el efecto generado por la cordillera no es muy notorio.

4. Si se observa el perfil, se puede notar claramente que la planta térmica no logra afectar una cantidad muy importante de la atmósfera, sólo los puntos cercanos alrededor de la fuente de calor. Esto se debe al tamaño de la planta, la que es bastante pequeña comparada al tamaño total de la atmósfera.

5. Anexo 1: Secciones del código de fuente

A continuación se muestran algunas secciones del código de fuente (en Python 2.7), ya que el código completo es muy extenso.

Se muestra la implementación de la clase *Litoral* con su constructor. Se excluyen algunos métodos.

```
1
2  # Creamos la clase Litoral con sus parametros
3  class Litoral(object):
4      def __init__(self, h, t1, t2, t3, t4, t5, it, alpha):
5
6          # Alpha (Bonus 3)
7          self.alpha = alpha
8          # Numero de iteraciones
9          self.it = it
10
11         # Tama o del perfil
12         self.alto = 2000  # metros
13         self.ancho = 4000  # metros
14
15         # Declaramos h y t
16         self.h = h
17         self.t1 = t1
18         self.t2 = t2
19         self.t3 = t3
20         self.t4 = t4
21         self.t5 = t5
22
23         # Tama o de la matriz
24         self.anchom = self.ancho / self.h
25         self.altom = self.alto / self.h
26
27         # Creamos la matriz
28         self.matrizCB = np.zeros((self.altom, self.anchom))  # matriz caso
       base
29         self.matriz1 = np.zeros((self.altom, self.anchom))   #
30         self.matriz2 = np.zeros((self.altom, self.anchom))   #
31         self.matriz3 = np.zeros((self.altom, self.anchom))   #
32         self.matriz4 = np.zeros((self.altom, self.anchom))   #
33         self.matriz5 = np.zeros((self.altom, self.anchom))   #
```

Lista de Códigos Fuente 1: Implementación de la clase *Litoral*

Se muestra la generación del terreno, el mar y la planta térmica. Se inicializan los puntos correspondientes a la planta con el valor que corresponde.

```
1
2      # Generacion del terreno
3      def perf(self):
4          # AGUA
5          porcentaje_raw = random.randint(30, 40)
6          limite = int(abs((porcentaje_raw / 100.0) * self.anchom))
7
```

```
        # T de la atmosfera en caso base
        j= self.altom
        for i in range(0, self.altom - (int(self.altom * 0.1))):
            self.matrizCB[i, :] = self.eq(j,self.t1)
            self.matriz1[i,:] = self.eq(j,self.t1)
            self.matriz2[i, :] = self.eq(j,self.t2)
            self.matriz3[i, :] = self.eq(j,self.t3)
            self.matriz4[i, :] = self.eq(j,self.t4)
            self.matriz5[i, :] = self.eq(j,self.t5)
            j -= 1

        ##rellenamos las ultimas filas (10%) con la condicion de borde (solo hasta el 30~40 %
        for i in range(1, limite + 1):
            for j in range(self.altom - (int(self.altom * 0.1)), self.altom):
                self.matrizCB[j, i - 1] = tagua(self.t1) #Caso base en t=0
                self.matriz1[j, i - 1] = tagua(self.t1)
                self.matriz2[j, i - 1] = tagua(self.t2)
                self.matriz3[j, i - 1] = tagua(self.t3)
                self.matriz4[j, i - 1] = tagua(self.t4)
                self.matriz5[j, i - 1] = tagua(self.t5)

        # MOUNTAINS
        prom = self.altom / 2

        # Para mountains
        alturas = np.ones(self.anchom - limite - (100/self.h))

        # Le damos altura inicial al bloque de monta as
        alturas *= prom
        if self.alpha == 1:
            indx1 = int((self.anchom - limite - (100/self.h))*0.1)
            indx2 = int((self.anchom - limite - (100/self.h))*0.2)
            indx3 = int((self.anchom - limite - (100/self.h))*0.3)
            indx4 = int((self.anchom - limite - (100/self.h))*0.4)
            indx5 = int((self.anchom - limite - (100/self.h))*0.5)
            indx6 = int((self.anchom - limite - (100/self.h))*0.6)
            indx7 = int((self.anchom - limite - (100/self.h))*0.7)
            indx8 = int((self.anchom - limite - (100/self.h))*0.8)
            indx9 = int((self.anchom - limite - (100/self.h))*0.9)

            # Tenemos las alturas "aleatorias", falta unirlas
            alturas[indx1] += randPercent()* random.random()*prom
            alturas[indx2] += randPercent()* random.random()*prom
            alturas[indx3] += randPercent()* random.random()*prom
            alturas[indx4] += randPercent()* random.random()*prom
            alturas[indx5] += randPercent()* random.random()*prom
            alturas[indx6] += randPercent()* random.random()*prom
            alturas[indx7] += randPercent()* random.random()*prom
            alturas[indx8] += randPercent()* random.random()*prom
            alturas[indx9] += randPercent()* random.random()*prom
```

```
58
59              alturas[0] = int(self.altom * 0.1)
60
61              # entre 0 y indx1
62              m1 = int(pendiente(0,indx1,alturas[0],alturas[indx1])) # no es
     int
63              j=1
64              for i in range(0,indx1+1):
65                  if i !=0:
66                      alturas[i] = m1*j + alturas[0]
67                      j+=1
68                  else:
69                      pass
70
71              # Calculamos la pendiente entre indx1 y indx2, y siguientes
72              m2 = int(pendiente(indx1,indx2,alturas[indx1],alturas[indx2]))
73              j=1
74              for i in range (indx1 +1, indx2+1):
75                  alturas[indx1+j] = m2*j + alturas[indx1]
76                  j+=1
77              m3 = int(pendiente(indx2, indx3, alturas[indx2], alturas[indx3
    ]))
78              j = 1
79              for i in range(indx2 + 1, indx3 + 1):
80                  alturas[indx2 + j] = m3 * j + alturas[indx2]
81                  j += 1
82
83              m4 = int(pendiente(indx3, indx4, alturas[indx3], alturas[indx4
    ]))
84              j = 1
85              for i in range(indx3 + 1, indx4 + 1):
86                  alturas[indx3 + j] = m4 * j + alturas[indx3]
87                  j += 1
88              m5 = int(pendiente(indx4, indx5, alturas[indx4], alturas[indx5
    ]))
89              j = 1
90              for i in range(indx4 + 1, indx5 + 1):
91                  alturas[indx4 + j] = m5 * j + alturas[indx4]
92                  j += 1
93
94              m6 = int(pendiente(indx5, indx6, alturas[indx5], alturas[indx6
    ]))
95              j = 1
96              for i in range(indx5 + 1, indx6 + 1):
97                  alturas[indx5 + j] = m6 * j + alturas[indx5]
98                  j += 1
99
100             m7 = int(pendiente(indx6, indx7, alturas[indx6], alturas[indx7
    ]))
101             j = 1
102             for i in range(indx6 + 1, indx7 + 1):
103                 alturas[indx6 + j] = m7 * j + alturas[indx6]
```

```
104                 j += 1
105
106             m8 = int(pendiente(indx7, indx8, alturas[indx7], alturas[indx8
    ]))
107             j = 1
108             for i in range(indx7 + 1, indx8 + 1):
109                 alturas[indx7 + j] = m8 * j + alturas[indx7]
110                 j += 1
111
112             m9 = int(pendiente(indx8, indx9, alturas[indx8], alturas[indx9
    ]))
113             j = 1
114             for i in range(indx8 + 1, indx9 + 1):
115                 alturas[indx8 + j] = m9 * j + alturas[indx8]
116                 j += 1
117
118             m10 = int(pendiente(indx9, self.anchom - limite -1, alturas[
    indx9], alturas[self.anchom - limite - (100/self.h) - 1]))
119             j = 1
120             for i in range(indx9 + 1, self.anchom - limite - (100/self.h))
    :
121                 alturas[indx9 + j] = m10 * j + alturas[indx9]
122                 j += 1
123         else:
124             pass
125
126         # Ingresamos mountains a la matriz (matrices)
127         j = 0
128         for i in range(limite + (100/self.h), self.anchom):
129             for k in range(int(alturas[j])-1):
130                 self.matriz1[self.altom - k - 1,i-1] = np.nan
131                 self.matriz2[self.altom - k - 1, i - 1] = np.nan
132                 self.matriz3[self.altom - k - 1, i - 1] = np.nan
133                 self.matriz4[self.altom - k - 1, i - 1] = np.nan
134                 self.matriz5[self.altom - k - 1, i - 1] = np.nan
135                 self.matrizCB[self.altom - k - 1, i - 1] = np.nan
136             j += 1
137
138         # Agregamos la planta
139         for i in range(0, (100/self.h)):
140             self.matriz1[self.altom - (int(self.altom * 0.1)), limite+i-1]
     = tempPlanta(self.t1)
141             self.matriz2[self.altom - (int(self.altom * 0.1)), limite + i
    - 1] = tempPlanta(self.t2)
142             self.matriz3[self.altom - (int(self.altom * 0.1)), limite + i
    - 1] = tempPlanta(self.t3)
143             self.matriz4[self.altom - (int(self.altom * 0.1)), limite + i
    - 1] = tempPlanta(self.t4)
144             self.matriz5[self.altom - (int(self.altom * 0.1)), limite + i
    - 1] = tempPlanta(self.t5)
145
146             for u in range(self.altom - (int(self.altom * 0.1))+1, self.
```

```
altom):
                self.matriz1[u,limite +i -1] = np.nan
                self.matriz2[u, limite + i - 1] = np.nan
                self.matriz3[u, limite + i - 1] = np.nan
                self.matriz4[u, limite + i - 1] = np.nan
                self.matriz5[u, limite + i - 1] = np.nan
                self.matrizCB[u, limite + i - 1] = np.nan
```

Lista de Códigos Fuente 2: Generación del terreno

Se muestra el algoritmo utilizado para iterar sobre las matrices. Se muestra solo sobre una matriz ya que el código es muy extenso (para ver la totalidad, ver main.py que se adjunta con este informe).

```

      # Iteraciones
      for _ in tqdm.tqdm(range(self.it)):
          for x in range(1, self.anchom-1):
              for y in range(1, self.altom-(int(self.altom * 0.1))):

                  # Hacemos un ciclo para iterar sobre las 6 matrices
distintas (los distintos t's)
                  for matrices in range(6):
                      if matrices == 0:
                          if np.isnan(self.matrizCB[y,x]):
                              pass
                          else:
                              A = self.matrizCB[y - 1, x]
                              B = self.matrizCB[y + 1, x]
                              C = self.matrizCB[y, x - 1]
                              D = self.matrizCB[y, x + 1]
                              if np.isnan(A):
                                  a = 15
                              else:
                                  a = A
                              if np.isnan(B):
                                  b = 15
                              else:
                                  b = B
                              if np.isnan(C):
                                  c = 15
                              else:
                                  c = C
                              if np.isnan(D):
                                  d = 15
                              else:
                                  d = D
                              self.matrizCB[y,x] = 0.25*(a+b+c+d)
                      if matrices == 1:
                      (...)
                      # Se utiliza el mismo algoritmo para matrices ==
2, 3, 4, 5 y 6
```

Lista de Códigos Fuente 3: Algoritmo de iteración para cada matriz (en este caso se muestra solo para la matriz matrizCB)

Para ejecutar el programa, se crea un objeto de tipo *Litoral* y se llama a sus métodos.

```
lit = Litoral(10, 0, 8, 12, 16, 20, 100, 1)
lit.perf()
lit.plot()
```

Lista de Códigos Fuente 4: Creación del objeto y ejecución