Biblioteca de generación y manejo de grafos

Escribir una biblioteca orientada a objetos, en Python 3.6, para describir y utilizar grafos. Debe, por lo menos contar con una clase llamada Grafo, una clase llamada Nodo y una clase llamada Arista. Asimismo, se deben realizar funciones para generar grafos con los siguientes modelos de generación:

1. **Modelo $G_{m,n}$ de malla.** Crear *m*n* nodos. Para el nodo $n_{i,j}$ crear una arista con el nodo $n_{i+1,j}$ y otra con el nodo $n_{i,j+1}$, para i<m y j<n

```
def grafoMalla(m, n, dirigido=False):
    """
    Genera grafo de malla
    :param m: número de columnas (> 1)
    :param n: número de filas (> 1)
    :param dirigido: el grafo es dirigido?
    :return: grafo generado
    """
```

2. **Modelo $G_{n,m}$ de Erdös y Rényi.** Crear *n* nodos y elegir uniformemente al azar *m* distintos pares de distintos vértices.

```
def grafoErdosRenyi(n, m, dirigido=False, auto=False):
    """
    Genera grafo aleatorio con el modelo Erdos-Renyi
    :param n: número de nodos (> 0)
    :param m: número de aristas (>= n-1)
    :param dirigido: el grafo es dirigido?
    :param auto: permitir auto-ciclos?
    :return: grafo generado
    """
```

3. **Modelo $G_{n,p}$ de Gilbert.** Crear *n* nodos y poner una arista entre cada par independiente y uniformemente con probabilidad *p.*

```
def grafoGilbert(n, p, dirigido=False, auto=False):
    """
    Genera grafo aleatorio con el modelo Gilbert
    :param n: número de nodos (> 0)
    :param p: probabilidad de crear una arista (0, 1)
    :param dirigido: el grafo es dirigido?
```

```
    :param auto: permitir auto-ciclos?
    :return: grafo generado
    """
```

4. **Modelo $G_{n,r}$ geográfico simple.** Colocar *n* nodos en un rectángulo unitario con coordenadas uniformes (o normales) y colocar una arista entre cada par que queda en distancia *r* o menor.

```
def grafoGeografico(n, r, dirigido=False, auto=False):
    """
    Genera grafo aleatorio con el modelo geográfico simple
    :param n: número de nodos (> 0)
    :param r: distancia máxima para crear un nodo (0, 1)
    :param dirigido: el grafo es dirigido?
    :param auto: permitir auto-ciclos?
    :return: grafo generado
    """
```

5. **Variante del modelo $G_{n,d}$ Barabási-Albert.** Colocar *n* nodos uno por uno, asignando a cada uno *d* aristas a vértices distintos de tal manera que la probabilidad de que el vértice nuevo se conecte a un vértice existente *v* es proporcional a la cantidad de aristas que *v* tiene actualmente - los primeros *d* vértices se conecta todos a todos.

```
def grafoBarabasiAlbert(n, d, dirigido=False, auto=False):
    """
    Genera grafo aleatorio con el modelo Barabasi-Albert
    :param n: número de nodos (> 0)
    :param d: grado máximo esperado por cada nodo (> 1)
    :param dirigido: el grafo es dirigido?
    :param auto: permitir auto-ciclos?
    :return: grafo generado
    """
```

6. **Modelo G_n Dorogovtsev-Mendes.** Crear 3 nodos y 3 aristas formando un triángulo. Después, para cada nodo adicional, se selecciona una arista al azar y se crean aristas entre el nodo nuevo y los extremos de la arista seleccionada.

```
def grafoBarabasiAlbert(n, dirigido=False):
    """
    Genera grafo aleatorio con el modelo Barabasi-Albert
    :param n: número de nodos (≥ 3)
    :param dirigido: el grafo es dirigido?
    :return: grafo generado
    """
```

La clase grafo debe contar con un método para guardar el grafo en un archivo con formato GraphViz (simple).

Entregar:

1. Link del repositorio en un servidor GIT (sugerido GitHub)
2. Archivos GV generados, 3 por cada modelo; uno con 30, otro con 100 y el tercero con 500 nodos.
3. Imágenes creadas con los grafos generados, 18 en total. Se sugiere utilizar Gephi (https://gephi.org/).