GRUPO BLR

CÁLCULO TOPOGRÁFICO CON PYTHON GUÍA DE PRÁCTICAS Y EJEMPLOS

ISBN: 978-9907-802-50-4

Javier Oswaldo Yunga Márquez

2026

CÁLCULO TOPOGRÁFICO CON PYTHON GUÍA DE PRÁCTICAS Y EJEMPLOS

Javier Oswaldo Yunga Márquez

AUTOR



UTMACH

Este libro ha sido debidamente examinado y valorado en la modalidad doble par ciego con fin de garantizar la calidad científica.

©Grupo Editorial BLR
Universidad Técnica de Machala
Riobamba – Ecuador
Correo: publicaciones@grupoblr.com
https://grupoblr.com/libros-investigo
REPOSITORIO



Yunga, J. (2026) Cálculo topográfico con Python guía de prácticas y ejemplos. Grupo Editorial BLR.

© Javier Oswaldo Yunga Márquez

Cálculo topográfico con Python guía de prácticas y ejemplos.

ISBN: 978-9907-802-50-4

El copyright promueve la libertad de expresión, protege la diversidad de ideas y conocimiento, además apoya la libre expresión. Se prohíbe de manera rigurosa la producción o el almacenamiento de esta publicación, ya sea en su totalidad o en parte, está estrictamente prohibido por ley, incluyendo el diseño de la portada, así como su difusión a través de cualquiera de sus medios, ya sean electrónicos, mecánicos, ópticos, de grabación o incluso de fotocopia, sin permiso de los propietarios de los derechos de autor.

FILIACIÓN DEL AUTOR

Javier Oswaldo Yunga Márquez

Universidad Técnica de Machala

Correo Electrónico: jyunga@utmachala.edu.ec

ORCID: https://orcid.org/0009-0005-2824-7122

PRÓLOGO

Este libro surge de la necesidad de integrar de manera práctica los fundamentos del cálculo topográfico con herramientas computacionales que permitan al estudiante analizar, comprobar y visualizar los procedimientos desarrollados durante su formación.

La experiencia docente en asignaturas relacionadas con topografía, ingenie-ría civil y tecnologías aplicadas ha mostrado la conveniencia de complementar los procedimientos tradicionales de cálculo con recursos que favorezcan la experimentación y la verificación de resultados. En este contexto, Python se incorpora como una herramienta de apoyo para desarrollar ejercicios, automatizar operaciones y representar información vinculada con problemas topográficos.

La obra ha sido concebida como una guía práctica orientada principalmente a estudiantes de ingeniería civil y áreas afines. Su propósito es acompañar el aprendizaje de los conceptos topográficos mediante ejemplos resueltos, ejercicios y fragmentos de código que relacionan los fundamentos matemáticos con su implementación computacional.

El uso de Python no pretende sustituir la comprensión de los procedimientos topográficos ni el criterio técnico necesario para su aplicación. Por el contrario, se propone como un recurso complementario que permite explorar diferentes escenarios, comprobar resultados y fortalecer la comprensión de los procesos de cálculo.

A lo largo del libro se presentan aplicaciones relacionadas con

planimetría, altimetría, modelos digitales del terreno, cálculo de volúmenes, GNSS, ajuste de observaciones, fotogrametría y sistemas de información geográfica. Cada tema combina fundamentos técnicos con ejemplos desarrollados mediante Python, procurando mantener una relación directa entre el procedimiento topográfico y su representación computacional.

Se espera que esta obra contribuya al desarrollo de competencias analíticas y tecnológicas mediante una aproximación práctica al cálculo topográfico, manteniendo como eje central la comprensión técnica de los procedimientos y el uso responsable de las herramientas computacionales.

ÍNDICE

PRÓLOGO ..... **i**

ÍNDICE ..... **iii**

ÍNDICE DE FIGURAS ..... **xii**

INTRODUCCIÓN ..... **xvii**

CAPÍTULO I **21**

1 INTRODUCCIÓN A PYTHON PARA APLICACIONES TOPOGRÁFICAS **21**

1.1 Tipos de datos y operadores 21

1.2 Control del flujo 22

1.3 Funciones y módulos 22

1.4 Problemas resueltos 23

1.4.1 Problema 1. Cálculo del área de un triángulo 23

1.4.2 Problema 2. Conversión de grados a radianes 23

1.4.3 Problema 3. Distancia euclidiana entre dos puntos 24

1.4.4 Problema 4. Promedio de elevaciones 24

1.4.5 Problema 5. Conversión de metros a pies 25

1.4.6 Problema 6. Clasificación didáctica de una pendiente 25

1.4.7 Problema 7. Cotas máximas y mínimas 26

1.4.8 Problema 8. Formateo de coordenadas 27

1.4.9 Problema 9. Ángulo de pendiente 27

1.4.10 Problema 10. Validación de cotas 28
1.5 Resumen 28
1.6 Problemas propuestos 29
CAPÍTULO II 30
2 PLANIMETRIA 30
2.1 Sistema de coordenadas 30
2.2 Levantamientos planimétricos 31
2.2.1 Cálculo de coordenadas a partir de azimut y distancia 32
2.3 Corrección de errores angulares y lineales 33
2.4 Problemas resueltos 35
2.4.1 Problema 1. Conversión de grados a radianes 35
2.4.2 Problema 2. Cálculo de coordenada final 35
2.4.3 Problema 3. Área de un triángulo 36
2.4.4 Problema 4. Clasificación didáctica de una pendiente 37
2.4.5 Problema 5. Distancia entre dos puntos 37
2.4.6 Problema 6. Error de cierre lineal 38
2.5 Resumen 39
2.6 Problemas propuestos 39
CAPÍTULO III 41
3 ALTIMETRÍA Y MODELO DIGITAL DEL TERRENO 41
3.1 Cotas y diferencias de nivel 41
3.2 Curvas de nivel 42

3.3 Modelo digital del terreno (MDT) 44
3.3.1 Construcción del MDT a partir de puntos 45
3.3.2 Visualización tridimensional del MDT 47
3.3.3 Análisis de pendientes 47
3.4 Problemas resueltos 48
3.4.1 Problema 1. Diferencia de nivel entre dos puntos 48
3.4.2 Problema 2. Perfil topográfico 49
3.4.3 Problema 3. Cálculo de pendiente entre dos puntos 50
3.4.4 Problema 4. Generación de un MDT a partir de puntos 51
3.4.5 Problema 5.Curvas de nivel de un MDT 52
3.4.6 Problema 6. Cálculo de pendientes en una malla 52
3.4.7 Problema 7. Clasificación didáctica del relieve 53
3.4.8 Problema 8. Visualización de puntos sobre el MDT 54
3.4.9 Problema 9. Exportación del MDT 55
3.4.10 Problema 10. Estadísticas básicas del MDT 55
3.5 Resumen 56
3.6 Problemas propuestos 57
CAPÍTULO IV 59
4 CÁLCULO DE VOLÚMENES EN TOPOGRAFÍA APLICADA 59
4.1 Fundamento geométrico y matemático 59
4.2 Método de secciones transversales 60

4.3 Método prismoidal 61
4.4 Método de cuadrícula 62
4.5 Volúmenes por diferencia de superficies 62
4.6 Cálculo de volúmenes con Python 63
4.6.1 Volumen por método prismoidal 63
4.6.2 Volumen por promedio de áreas extremas 64
4.6.3 Volumen por diferencia de MDT 64
4.6.4 Visualización de cortes y rellenos 65
4.6.5 Histograma de diferencias de elevación 65
4.7 Comparación de métodos 65
4.8 Problemas resueltos 66
4.8.1 Problema 1. Volumen entre dos secciones 66
4.8.2 Problema 2. Volumen prismoidal 67
4.8.3 Problema 3. Volumen por cuadrícula 68
4.8.4 Problema 4. Corte por diferencia entre MDT 68
4.8.5 Problema 5. Separación entre corte y relleno 70
4.8.6 Problema 6. Histograma de diferencias 70
4.8.7 Problema 7. Volumen acumulado entre secciones 71
4.8.8 Problema 8. Mapa de diferencias de elevación 72
4.8.9 Problema 9. Cálculo del volumen de una rampa 72
4.8.10 Problema 10. Estadísticas básicas de diferencias 73
4.9 Resumen 74

4.10 Problemas propuestos 74

CAPÍTULO V 77

5 GNSS y georreferenciación en topografía con Python 77

5.1 Fundamentos de los sistemas GNSS 77

5.2 Sistemas de coordenadas y georreferenciación 79

5.2.1 Coordenadas geográficas 79

5.2.2 Coordenadas UTM 80

5.3 Transformación entre sistemas de coordenadas 80

5.4 Manejo de datos espaciales con GeoPandas 82

5.5 Visualización de trayectorias GNSS 83

5.6 Errores en posicionamiento GNSS 84

5.7 Problemas resueltos 85

5.7.1 Problema 1. Conversión de coordenadas geográficas a UTM ... 85

5.7.2 Problema 2. Determinación de la zona UTM 85

5.7.3 Problema 3. Tiempo de viaje de una señal GNSS 86

5.7.4 Problema 4. Conversión de grados decimales a sexagesimales . 87

5.7.5 Problema 5. Determinación del hemisferio 87

5.7.6 Problema 6. Cálculo de PDOP y GDOP 88

5.7.7 Problema 7. Desplazamiento horizontal relativo 89

5.7.8 Problema 8. Aplicación de una corrección local de coordenadas 90

5.7.9 Problema 9. Diferencia de altura entre dos puntos GNSS 91

5.7.10 Problema 10. Velocidad media de un recorrido 91
5.8 Resumen 92
5.9 Problemas propuestos 93
CAPÍTULO VI 95
6 AJUSTE DE OBSERVACIONES TOPOGRÁFICAS POR MÍNIMOS CUADRADOS CON PYTHON 95
6.1 Fundamento matemático del método 96
6.2 Modelo funcional y modelo estocástico 97
6.3 Ajuste por mínimos cuadrados con Python 98
6.4 Interpretación de resultados 99
6.5 Matriz de covarianzas y precisión del ajuste 100
6.6 Aplicación a una red topográfica 2D simplificada 101
6.7 Problemas resueltos 103
6.7.1 Problema 1. Ajuste de una distancia observada varias veces 103
6.7.2 Problema 2. Ajuste ponderado de una dirección 103
6.7.3 Problema 3. Cálculo de la varianza de referencia a posteriori 104
6.7.4 Problema 4. Ajuste de coordenadas planas repetidas 105
6.7.5 Problema 5. Error de cierre lineal como control previo 106
6.7.6 Problema 6. Compensación angular simple por reparto uniforme 106
6.7.7 Problema 7. Cálculo inverso entre dos puntos 107
6.7.8 Problema 8. Matriz de covarianzas de las incógnitas 108

6.7.9 Problema 9. Área de un polígono por coordenadas 109

6.7.10 Problema 10. Verificación de perpendicularidad entre direcciones..... 109

6.8 Resumen..... 110

6.9 Problemas propuestos 111

CAPÍTULO VII 112

7 FOTOGRAMETRÍA, NUBES DE PUNTOS Y VISUALIZACIÓN 3D CON PYTHON 112

7.1 Principios básicos de la fotogrametría 112

7.2 Nubes de puntos 114

7.3 Simulación de una nube de puntos en Python..... 115

7.4 Lectura de nubes de puntos reales..... 115

7.5 Visualización avanzada de nubes de puntos 116

7.6 Ortomosaicos y modelos digitales 117

7.7 Problemas resueltos..... 118

7.7.1 Problema 1. Número total de puntos en una nube simulada.... 118

7.7.2 Problema 2. Cálculo de elevación media de una nube simulada 118

7.7.3 Problema 3. Altura máxima y mínima de una nube..... 119

7.7.4 Problema 4. Cálculo del centroide de una nube de puntos 119

7.7.5 Problema 5. Filtrado por elevación 120

7.7.6 Problema 6. Visualización 3D de una nube de puntos..... 120

7.7.7 Problema 7. Generación de un perfil a partir de puntos..... 120
7.7.8 Problema 8. Estimación simple de densidad de puntos 121
7.7.9 Problema 9. Exportación de una nube simulada a CSV 122
7.7.10 Problema 10. Evaluación de dispersión vertical 123
7.8 Resumen 123
7.9 Problemas propuestos 123
CAPÍTULO VIII 125
8 SIG Y ANÁLISIS ESPACIAL CON PYTHON 125
8.1 Conceptos fundamentales..... 125
8.2 Lectura de datos vectoriales con GeoPandas 126
8.3 Análisis espacial: buffers e intersecciones 127
8.4 Análisis ráster con Rasterio..... 128
8.5 Cálculo de pendientes y orientación 129
8.6 Mapas interactivos con Folium 129
8.7 Problemas resueltos..... 130
8.7.1 Problema 1. Cálculo del área de una cuenca..... 130
8.7.2 Problema 2. Buffer alrededor de un río..... 130
8.7.3 Problema 3. Intersección con cultivos 131
8.7.4 Problema 4. Lectura de MDT y cálculo de pendiente..... 131
8.7.5 Problema 5. Clasificación del relieve..... 131
8.7.6 Problema 6. Exportación a GeoJSON..... 133
8.7.7 Problema 7. Visualización con Folium 133

8.7.8 Problema 8. Conteo de parcelas por categoría 133

8.7.9 Problema 9. Extracción de valores de ráster 133

8.7.10 Problema 10. Unión espacial entre capas 134

8.8 Resumen 134

8.9 Problemas propuestos 135

RESUMEN FINAL 137

CONCLUCIONES 139

BIBLIOGRAFÍA 141

ÍNDICE DE FIGURAS

Figura 1. Código 1.1: Tipos de datos básicos. 22
Figura 2. Código 1.2: Condicionales y bucles. 22
Figura 3. Código 1.3: Función simple. 23
Figura 4. Código 1.4: Área de un triángulo. 23
Figura 5. Código 1.5: Conversión de grados a radianes. 24
Figura 6. Código 1.6: Distancia euclidiana. 24
Figura 7. Código 1.7: Promedio de elevaciones. 25
Figura 8. Código 1.8: Conversión de unidades. 25
Figura 9. Código 1.9: Clasificación didáctica de una pendiente. .. 26
Figura 10. Código 1.10: Cotas máximas y mínimas. 26
Figura 11. Código 1.11: Formateo de coordenadas. 27
Figura 12. Código 1.12: Ángulo de pendiente. 27
Figura 13. Código 1.13: Validación de cotas. 28
Figura 14. Código 2.1: Cálculo de coordenadas con azimut y distancia. 33
Figura 15. Código 2.2: Grados a radianes. 35
Figura 16. Código 2.3: Coordenada final. 36
Figura 17. Código 2.4: Área de un triángulo. 36
Figura 18. Código 2.5: Clasificación didáctica de una pendiente. 37
Figura 19. Código 2.6: Distancia euclidiana. 38
Figura 20. Código 2.7: Error de cierre lineal. 38
Figura 21. Código3.1:Interpolaciónycurvasdenivel. 44
Figura 22. Código3.2:GeneracióndeMDTinterpolado. 46
Figura 23. Código 3.3: Visualización 3D del MDT. 47
Figura 24. Código 3.4: Análisis de pendientes. 48

Figura 25. Código 3.5: Diferencia de nivel. 49
Figura 26. Código 3.6: Perfil topográfico. 50
Figura 27. Código 3.7: Pendiente entre dos puntos. 51
Figura 28. Código3.8:Generación de un MDT. 52
Figura 29. Código3.9:Curvas de nivel a partir de un MDT. 52
Figura 30. Código 3.10: Mapa de pendiente. 53
Figura 31. Código 3.11: Clasificación didáctica del relieve. 54
Figura 32. Código 3.12:Puntos sobre el MDT. 55
Figura 33. Código 3.13: Exportación del MDT. 55
Figura 34. Código 3.14: Estadísticas básicas del MDT. 55
Figura 35. Código 4.1: Volumen por método prismoidal. 63
Figura 36. Código 4.2: Volumen por secciones transversales. 64
Figura 37. Código 4.3: Volumen por diferencia de MDT. 64
Figura 38. Código 4.4: Mapa de cortes y rellenos. 65
Figura 39. Código 4.5: Histograma de diferencias de elevación. .. 65
Figura 40. Código 4.6: Volumen entre dos secciones. 67
Figura 41. Código 4.7: Volumen por método prismoidal. 67
Figura 42. Código 4.8: Volumen por cuadrícula. 68
Figura 43. Código 4.9: Diferencia entre MDT. 69
Figura 44. Código 4.10: Separación de corte y relleno. 70
Figura 45. Código4.11:Histogramadediferencias. 71
Figura 46. Código4.12:Volumenacumuladoporsecciones. 71
Figura 47. Código 4.13: Mapa de diferencias de elevación. 72
Figura 48.Código4.14:Volumen de una rampa. 73
Figura 49. Código4.15:Estadísticas de diferencias. 73
Figura 50. Código 5.1: Conversión de WGS 84 a UTM zona 17 Sur. 81

Figura 51. Código 5.2: Creación de un Geo Data Frame con coordenadas GNSS. 82
Figura 52. Código 5.3: Transformación de GeoDataFrame a UTM. 82
Figura 53. Código 5.4: Visualización de trayectoria GNSS. 83
Figura 54. Código 5.5: Coordenadas geográficas a UTM 85
Figura 55. Código 5.6: Determinación de la zona UTM. 86
Figura 56. Código 5.7: Tiempo de viaje de una señal GNSS. 87
Figura 57. Código 5.8: Conversión de decimal a sexagesimal. 87
Figura 58. Código 5.9: Determinación del hemisferio. 88
Figura 59. Código 5.10: Cálculo de PDOP y GDOP. 89
Figura 60. Código 5.11: Desplazamiento horizontal relativo. 89
Figura 61. Código 5.12: Corrección local de coordenadas. 90
Figura 62. Código 5.13: Diferencia de altura GNSS. 91
Figura 63. Código 5.14: Velocidad media de un recorrido. 92
Figura 64. Código 6.1: Ajuste clásico por mínimos cuadrados. 99
Figura 65. Código 6.2: Ajuste simplificado de dos coordenadas. 102
Figura 66. Código 6.3: Ajuste de una distancia repetida. 103
Figura 67. Código 6.4: Ajuste ponderado de una dirección. 104
Figura 68. Código 6.5: Varianza de referencia a posteriori. 105
Figura 69. Código 6.6: Ajuste de coordenadas planas. 105
Figura 70. Código 6.7: Magnitud del error de cierre. 106
Figura 71. Código 6.8: Compensación angular simple. 107
Figura 72. Código 6.9: Cálculo inverso entre dos puntos. 108
Figura 73. Código 6.10: Matriz de covarianzas de las incógnitas. 108
Figura 74. Código 6.11: Área de un polígono por coordenadas. . 109

Figura 75. Código 6.12: Verificación de perpendicularidad. 110
Figura 76. Código 7.1: Generar y visualizar una nube de puntos simulada..... 116
Figura 77. Código 7.2: Lectura de archivo LAS con laspy. 116
Figura 78. Código 7.3: Visualizacion 3D de nube de puntos con PyntCloud..... 116
Figura 79. Código7.4: Número de puntos en una nube simulada. 118
Figura 80. Código7.5:Elevacion media de una nube simulada. .. 118
Figura 81. Código 7.6: Altura máxima y mínima de una nube.. 119
Figura 82. Código 7.7: Centroide de una nube de puntos. 119
Figura 83. Código 7.8: Filtrado por elevación. 120
Figura 84. Código 7.9: Visualización 3D de una nube simulada. 121
Figura 85. Código 7.10: Perfil de elevaciones desde puntos. 121
Figura 86. Código 7.11: Densidad de puntos. 122
Figura 87. Código 7.12: Exportación de nube a CSV. 122
Figura 88. Código 7.13: Dispersión vertical. 122
Figura 89. Código 8.1: Cargar shapefile y mostrar información.126
Figura 90. Código 8.2: Crear un buffer de 500 m alrededor de caminos. 127
Figura 91. Código 8.3: Intersección con zonas protegidas..... 127
Figura 92.Código 8.4: Leer raster y graficar mapa de elevacion. 128
Figura 93. Código 8.5: Calculo de pendiente con NumPy. 128
Figura 94. Código 8.6: Mapa web con puntos GNSS. 130
Figura 95. Código 8.7: Area de una cuenca. 130
Figura 96. Código 8.8: Buffer de rio. 131

Figura 97. Código 8.9: Intersección con cultivos. 131
Figura 98. Código 8.10: Mapa de pendientes. 132
Figura 99. Código 8.11: Clasificación de relieve. 132
Figura 100. Código 8.12: Exportar puntos a GeoJSON. 132
Figura 101. Código 8.13: Mapa web con folium. 133
Figura 102. Código 8.14: Conteo por categoría. 134
Figura 103. Código 8.15: Extracción de elevaciones. 134
Figura 104. Código 8.16: Union espacial. 135

INTRODUCCIÓN

La topografía comprende procedimientos de medición, cálculo y representación que permiten determinar la posición relativa de puntos sobre el terreno y generar información necesaria para diferentes aplicaciones de la ingeniería. El desarrollo de estos procedimientos requiere comprender fun-damentos geométricos, trigonométricos y estadísticos, así como interpretar adecuadamente los resultados obtenidos.

En la formación académica, muchos cálculos topográficos se desarrollan inicialmente mediante procedimientos manuales con el propósito de com-prender sus fundamentos. Sin embargo, a medida que aumenta la cantidad de datos y la complejidad de los problemas, resulta conveniente incorpo-rar herramientas computacionales que permitan automatizar operaciones repetitivas, comprobar resultados y representar la información obtenida.

En este contexto, Python constituye una herramienta que puede complementar el aprendizaje y la aplicación del cálculo topográfico. Su utilización permite implementar procedimientos matemáticos, procesar conjuntos de datos y desarrollar representaciones gráficas mediante códigos que pueden ser examinados, modificados y ejecutados por el estudiante. El uso de la programación se plantea en esta obra como un recurso complementario y no como sustituto de la comprensión de los fundamentos y del criterio técnico requerido en la práctica topográfica.

Objetivo general

Proporcionar a los estudiantes una guía para comprender y aplicar conceptos y procedimientos del cálculo topográfico mediante el uso complementario de Python para el desarrollo de cálculos, la experimentación, la visualiza-ción y la resolución de problemas relacionados con diferentes áreas de la topografía.

Alcance de la obra

El libro está orientado principalmente a estudiantes de ingeniería civil y áreas afines que requieren aplicar procedimientos de cálculo topográfico y explorar su implementación mediante herramientas computacionales.

Los ejemplos desarrollados combinan fundamentos técnicos, expresiones matemáticas y código Python. Esta integración permite observar la relación entre el procedimiento de cálculo y su implementación computacional, man-teniendo como principio que la interpretación de los resultados depende del conocimiento técnico y de las condiciones particulares de cada problema.

La obra aborda aplicaciones relacionadas con programación para cálculos topográficos, planimetría, altimetría, modelos digitales del terreno, cálcu-lo de volúmenes, sistemas GNSS, ajuste de observaciones, fotogrametría, nubes de puntos y sistemas de información geográfica.

Estructura del libro

El contenido se organiza en ocho capítulos:

Capítulo 1. Introducción a Python para aplicaciones topográficas. Presenta conceptos básicos de Python y su aplicación en operaciones y problemas relacionados con el cálculo topográfico.

Capítulo 2. Planimetría. Aborda procedimientos para el cálculo y tratamiento de información planimétrica mediante ejemplos y aplica-ciones desarrolladas con Python.

Capítulo 3. Altimetría y modelo digital del terreno. Desarrolla procedimientos relacionados con diferencias de nivel, pendientes y representación de información altimétrica y del terreno.

Capítulo 4. Cálculo de volúmenes en topografía aplicada. Presenta procedimientos para determinar volúmenes a partir de información topográfica mediante diferentes métodos de cálculo.

Capítulo 5. GNSS y georreferenciación en topografía con Python. Introduce conceptos y procedimientos relacionados con posiciona-miento, coordenadas y procesamiento de información GNSS.

Capítulo 6. Ajuste de observaciones topográficas por mínimos cuadrados con Python. Desarrolla fundamentos y aplicaciones para el tratamiento y ajuste de observaciones topográficas.

Capítulo 7. Fotogrametría, nubes de puntos y visualización 3D con Python. Presenta aplicaciones relacionadas con información fo-togramétrica, procesamiento de nubes de puntos y representación tridimensional.

Capítulo 8. SIG y análisis espacial con Python. Aborda procedimientos básicos de análisis espacial y manejo de información geográ-fica mediante herramientas computacionales.

Cada capítulo combina fundamentos conceptuales con ejemplos, problemas y códigos orientados a relacionar los procedimientos topográficos con su implementación computacional. De esta manera, el lector puede utilizar los ejemplos como apoyo para comprender los cálculos, comprobar resultados y explorar variaciones de los problemas planteados.

CAPÍTULO I

1 INTRODUCCIÓN A PYTHON PARA APLICACIONES TOPOGRÁFICAS

Python es un lenguaje de programación moderno, interpretado y de alto nivel, que ha alcanzado una amplia difusión debido a su sintaxis clara, legi-bilidad y versatilidad. En el ámbito de la topografía, puede emplearse para el procesamiento de datos, la automatización de cálculos y la visualización de información y modelos del terreno (Van Rossum & Drake, 2009).

En las geociencias, Python dispone de bibliotecas científicas como numpy, pandas, matplotlib y scipy, que permiten realizar análisis numéricos, manipular datos de manera eficiente y generar representaciones gráficas en dos y tres dimensiones (McKinney, 2022; Hunter, 2007; Virtanen et al., 2020).

Este capítulo tiene como objetivo introducir al lector en los fundamentos de la sintaxis de Python y en su aplicación a tareas relacionadas con la topografía, como el cálculo de distancias, la transformación de coordenadas y la representación de información altimétrica (Kavanagh & Slattery, 2014; Wolf & Ghilani, 2012).

1.1 Tipos de datos y operadores

Python admite distintos tipos de datos, entre ellos los numéricos, como int y float; las cadenas de texto, representadas mediante str; los valores booleanos, mediante bool; y estructuras compuestas como listas, tuplas

y diccionarios. Asimismo, permite realizar operaciones aritméticas, lógicas y de comparación.

```
# Números
x = 10
y = 3.5

# Cadenas de texto
nombre = "Estación A"

# Booleanos
activo = True
```

Figura 1. Código 1.1: Tipos de datos básicos.

1.2 Control del flujo

Las estructuras condicionales if, elif y else, junto con los bucles for y while, permiten controlar el flujo de ejecución de un programa y establecer diferentes acciones según las condiciones definidas.

```
for i in range(1, 4):
if i % 2 == 0:
print("Par:", i)
else:
print("Impar:", i)
```

Figura 2. Código 1.2: Condicionales y bucles.

1.3 Funciones y módulos

Python permite definir funciones personalizadas para organizar el código, reutilizar procedimientos y facilitar su mantenimiento. Además, mediante la importación de módulos se pueden incorporar funciones adicionales para realizar cálculos matemáticos y otras operaciones específicas.

```python
def distancia(x1, x2):
return abs(x2 - x1)

print(distancia(100, 130))    # Resultado: 30
```

Figura 3. Código 1.3: Función simple.

1.4 Problemas resueltos

1.4.1 Problema 1. Cálculo del área de un triángulo

Dado un triángulo con una base de 25 metros y una altura de 12 metros, calcular su área utilizando Python.

Solución:

$$A = 1/_2 \times \text{base} \times \text{altura}$$

```python
base = 25
altura = 12

area = 0.5 * base * altura

print(f"Área del triángulo: {area:.2f} m^2")
```

Figura 4. Código 1.4: Área de un triángulo.

1.4.2 Problema 2. Conversión de grados a radianes

Convertir un ángulo de 45◦ a radianes.

Solución:

La conversión puede realizarse mediante la función math.radians() del módulo math.

```python
import math

grados = 45
radianes = math.radians(grados)

print(f"{grados} grados equivalen a {radianes:.4f} radianes")
```

Figura 5. Código 1.5: Conversión de grados a radianes.

1.4.3 Problema 3. Distancia euclidiana entre dos puntos

Calcular la distancia entre los puntos $A(30, 50)$ y $B(80, 90)$.

Solución:

La distancia entre dos puntos en el plano se determina mediante:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```python
import math

x1, y1 = 30, 50
x2, y2 = 80, 90

distancia = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

print(f"Distancia entre A y B: {distancia:.2f} m")
```

Figura 6. Código 1.6: Distancia euclidiana.

1.4.4 Problema 4. Promedio de elevaciones

Obtener el promedio de las elevaciones 220, 224, 218, 230 y 227 metros.

Solución:

```
elevaciones = [220, 224, 218, 230, 227]

promedio = sum(elevaciones) / len(elevaciones)

print(f"Elevación promedio: {promedio:.2f} m")
```

Figura 7. Código 1.7: Promedio de elevaciones.

1.4.5 Problema 5. Conversión de metros a pies

Convertir 100 metros a pies, considerando la equivalencia:

1 m = 3.28084 pies

Solución:

```
metros = 100
pies = metros * 3.28084

print(f"{metros} metros = {pies:.2f} pies")
```

Figura 8. Código 1.8: Conversión de unidades.

1.4.6 Problema 6. Clasificación didáctica de una pendiente

Para este ejercicio se adopta, únicamente con fines didácticos, la siguiente clasificación de pendientes:

- **Suave:** pendiente menor que 0.08.
- **Media:** pendiente entre 0.08 y 0.15.
- **Fuerte:** pendiente mayor que 0.15.

Clasificar una pendiente de 0.12 de acuerdo con estos intervalos. Estos límites se utilizan únicamente para ilustrar el funcionamiento de las

estructuras condicionales y no deben interpretarse como una clasificación normativa de pendientes.

Solución:

```
pendiente = 0.12

if pendiente < 0.08:
clasificacion = "Suave"
elif pendiente <= 0.15:
clasificacion = "Media"
else:
clasificacion = "Fuerte"

print(
f"Pendiente: {pendiente:.2f} -> "
f"Clasificación: {clasificacion}"
)
```

Figura 9. Código 1.9: Clasificación didáctica de una pendiente.

1.4.7 Problema 7. Cotas máximas y mínimas

Obtener la cota máxima y la cota mínima del siguiente conjunto de valores:

$$435.6,\ 440.2,\ 437.8,\ 439.1,\ 438.5$$

Solución:

```
cotas = [435.6, 440.2, 437.8, 439.1, 438.5]

print(f"Cota máxima: {max(cotas)} m")
print(f"Cota mínima: {min(cotas)} m")
```

Figura 10. Código 1.10: Cotas máximas y mínimas.

1.4.8 Problema 8. Formateo de coordenadas

Mostrar de forma ordenada la identificación y las coordenadas de tres puntos topográficos.

Solución:

```
puntos = [
("P1", 100, 200),
("P2", 105, 198),
("P3", 110, 202)
]

for nombre, x, y in puntos:
print(f"{nombre}: X={x} m, Y={y} m")
```

Figura 11. Código 1.11: Formateo de coordenadas.

1.4.9 Problema 9. Ángulo de pendiente

Calcular el ángulo, expresado en grados, correspondiente a una pendiente de 0.2.

Solución: Si la pendiente se expresa como la razón entre el desnivel y la distancia horizontal, el ángulo se obtiene mediante:

$$\theta = \arctan(m)$$

donde m representa la pendiente.

```
import math

pendiente = 0.2
angulo = math.degrees(math.atan(pendiente))

print(f"Ángulo de pendiente: {angulo:.2f} grados")
```

Figura 12. Código 1.12: Ángulo de pendiente.

1.4.10 Problema 10. Validación de cotas

Verificar si todas las cotas del conjunto se encuentran dentro del intervalo comprendido entre 200 y 300 metros.

Solución:

```
        cotas = [245, 260, 275, 210, 295]

        validas = all(200 <= c <= 300 for c in cotas)

        print(f"Todas las cotas son válidas: {validas}")
```

Figura 13. Código 1.13: Validación de cotas.

1.5 Resumen

En este capítulo se presentaron los fundamentos de Python necesarios para desarrollar aplicaciones sencillas relacionadas con el cálculo topográfico. Se revisaron los principales tipos de datos, las estructuras de control, la definición de funciones y el uso de módulos. Los problemas resueltos mostraron la aplicación de estos elementos en cálculos de áreas, distancias, conversiones angulares y de unidades, procesamiento de elevaciones, clasificación didáctica de pendientes y manejo de coordenadas.

Estos conocimientos proporcionan una base para los capítulos siguientes, en los que se desarrollan procedimientos topográficos de mayor complejidad mediante el uso de Python

1.6 Problemas propuestos

1. Identificación de tipos de datos. Defina variables que representen el nombre de un punto topográfico, su cota y su estado de activación. Muestre el valor y el tipo de dato correspondiente a cada variable.

2. Condicional simple. Escriba un programa que determine si la cota de un punto se encuentra por encima, por debajo o exactamente en un nivel de referencia de 100 m.

3. Bucle de control. Desarrolle una rutina que calcule la diferencia de altura entre una estación de referencia y cinco puntos consecutivos mediante un bucle for.

4. Función de distancia. Defina inicialmente una función que reciba dos coordenadas sobre el eje X y calcule la distancia entre ellas. Posteriormente, amplíe la función para recibir las coordenadas (X, Y) de dos puntos y calcular su distancia bidimensional.

5. Aplicación de pendientes. Considere tres puntos consecutivos con cotas de 100.2 m, 102.5 m y 101.8 m. Defina distancias horizontales entre los puntos y escriba un programa que calcule las pendientes de cada tramo y una pendiente media para el conjunto. Indique claramente el criterio utilizado para calcular dicha media.

6. Análisis de cotas. Lea una lista de cotas y determine el valor máximo y mínimo. Muestre también la posición que ocupa cada uno dentro de la lista.

CAPÍTULO II

2 PLANIMETRIA

La planimetría es la rama de la topografía encargada de estudiar, medir y representar la proyección horizontal de los elementos del terreno sobre un plano de referencia. Su objetivo principal es determinar la posición relativa de los puntos mediante coordenadas planas (X, Y), sin considerar en esta etapa la componente altimétrica. Esta disciplina constituye una base para la elaboración de planos, el replanteo de obras, el diseño vial, la delimitación predial y el control geométrico en proyectos de ingeniería civil, agrícola, minera y ambiental (Wolf & Ghilani, 2012).

En la práctica topográfica, las coordenadas planas pueden obtenerse mediante levantamientos realizados con estación total, teodolito, cinta métrica, receptores GNSS u otros instrumentos y métodos de medición. La precisión de los resultados depende no solo de las características del instrumento utilizado, sino también de la metodología de observación, la geometría del levantamiento, las condiciones de campo, la estabilidad de los puntos de control, el procesamiento de los datos y el ajuste de los errores asociados a las mediciones (Kavanagh & Slattery, 2014).

2.1 Sistema de coordenadas

Para la representación planimétrica se utiliza un sistema cartesiano bidimensional en el que cada punto queda definido por una pareja ordenada (X, Y). En topografía, estas coordenadas suelen asociarse convencionalmente con las direcciones Este y Norte, aunque también

pueden expresarse en sistemas locales definidos para un proyecto o en sistemas georreferenciados vinculados a marcos de referencia nacionales o globales.

La calidad de las coordenadas obtenidas depende de factores como la precisión de las observaciones, la metodología empleada, la geometría del levantamiento, la estabilidad de los puntos de control, el tratamiento de los errores y el sistema de referencia adoptado. En proyectos de ingeniería, la correcta definición del sistema de coordenadas es fundamental para garantizar la compatibilidad entre las etapas de levantamiento, diseño y replanteo.

2.2 Levantamientos planimétricos

Los métodos clásicos de levantamiento planimétrico incluyen, entre otros, la radiación, la intersección y las poligonales.

- **Radiación:** consiste en determinar la posición de puntos desde una estación conocida mediante la medición de direcciones y distancias.

- **Intersección:** permite determinar la posición de un punto a partir del cruce de visuales o de distancias observadas desde estaciones conocidas.

- **Poligonales:** se basan en la medición sucesiva de ángulos y distancias entre vértices consecutivos para calcular las coordenadas de una serie de puntos enlazados.

Entre estos métodos, las poligonales tienen una aplicación importante en topografía, ya que permiten enlazar puntos de control, levantar detalles y establecer redes de apoyo para proyectos de ingeniería. Para su tratamiento matemático, es necesario calcular incrementos de coordenadas a partir de una dirección y una distancia conocidas.

2.2.1 Cálculo de coordenadas a partir de azimut y distancia

Cuando la dirección se expresa mediante un azimut medido desde el norte en sentido horario, los incrementos de coordenadas se calculan mediante:

$$\Delta X = d \sin(\alpha) \text{ y} \qquad \Delta Y = d \cos(\alpha)$$

donde:

d es la distancia horizontal;

α es el azimut;

ΔX es el incremento en la coordenada Este;

ΔY es el incremento en la coordenada Norte.

Si el azimut se expresa en grados, debe convertirse previamente a radianes antes de utilizar las funciones trigonométricas de Python.

Las coordenadas del punto final se obtienen mediante:

$$Xf = Xi + \Delta X \text{y} \qquad Yf = Yi + \Delta Y$$

```python
import math

def coordenada_final(xi, yi, azimut_grados, distancia):
azimut_rad = math.radians(azimut_grados)

xf = xi + distancia * math.sin(azimut_rad)
yf = yi + distancia * math.cos(azimut_rad)

return xf, yf

x2, y2 = coordenada_final(
1000.25,
2000.80,
45.75,
100.40
)

print(f"Nueva coordenada: X={x2:.3f}, Y={y2:.3f}")
```

Figura 14. Código 2.1: Cálculo de coordenadas con azimut y distancia.

Este procedimiento es aplicable cuando la dirección se expresa mediante azimut. Si se utilizan rumbos cuadrantales, debe realizarse previamente la conversión correspondiente al azimut antes de aplicar estas relaciones trigonométricas.

2.3 Corrección de errores angulares y lineales

En los levantamientos mediante poligonales, las observaciones están suje-tas a errores asociados con el instrumento, el observador, las condiciones ambientales y del terreno, así como con el procesamiento de los datos. Por ello, una vez medidas las direcciones y las distancias, es necesario verificar los cierres geométricos del levantamiento y, cuando corresponda, aplicar un procedimiento de ajuste.

En una poligonal cerrada, la suma teórica de los ángulos interiores está dada por:

$$\sum \beta \text{teórica} = (n - 2) \times 180^{\circ}$$

donde n representa el número de vértices de la poligonal.

La diferencia entre la suma de los ángulos observados y la suma teórica constituye el error angular de cierre:

$$f\beta = \sum \beta \text{observada} - \sum \beta \text{teórica}$$

Después de realizar el ajuste angular, se calculan los incrementos de coordenadas correspondientes a cada lado de la poligonal. En una poligonal cerrada ideal, la suma de estos incrementos debería ser igual a cero. En la práctica, las discrepancias se expresan mediante los errores de cierre en coordenadas:

$$fx = \sum \Delta X \qquad \text{y} \qquad fy = \sum \Delta Y$$

A partir de estas componentes se obtiene el error de cierre lineal:

$$e = \sqrt{f\,2 + f\,2}$$

El error de cierre puede distribuirse entre los lados de la poligonal median-te diferentes procedimientos de ajuste. Uno de los métodos clásicos es la regla de Bowditch, también denominada regla de la brújula, en la cual las correcciones de los incrementos de coordenadas se distribuyen proporcionalmente a las longitudes de los lados. Su aplicación permite obtener un conjunto de coordenadas ajustadas que satisfacen el cierre geométrico de la poligonal; sin embargo, cuando se requiere un tratamiento estadístico más riguroso de las observaciones pueden emplearse métodos de ajuste por mínimos cuadrados (Ghilani, 2021).

En trabajos que requieren un tratamiento estadístico más riguroso de las observaciones, pueden emplearse métodos de ajuste por mínimos cuadrados, los cuales se desarrollan posteriormente en este libro.

2.4 Problemas resueltos

2.4.1 Problema 1. Conversión de grados a radianes

Convertir 45.35◦ a radianes.

Solución:

La conversión angular es necesaria porque las funciones trigonométricas del módulo math de Python trabajan con ángulos expresados en radianes.

```
import math

grados = 45.35
radianes = math.radians(grados)

print(f"{grados:.2f} grados = {radianes:.4f} rad")
```

Figura 15. Código 2.2: Grados a radianes.

2.4.2 Problema 2. Cálculo de coordenada final

Dado un punto inicial $A(1000.25, 2000.80)$, un azimut de 45.75◦ y una distancia horizontal de 100.40 m, calcular las coordenadas del punto final.

Solución:

Se aplican las relaciones:

$$\Delta X = d \sin(\alpha) \text{ y} \qquad \Delta Y = d \cos(\alpha)$$

```python
import math

def coordenada_final(xi, yi, azimut_grados, distancia):
azimut_rad = math.radians(azimut_grados)

xf = xi + distancia * math.sin(azimut_rad)
yf = yi + distancia * math.cos(azimut_rad)

return xf, yf

x2, y2 = coordenada_final(
1000.25,
2000.80,
45.75,
100.40
)

print(f"X final = {x2:.3f} m")
print(f"Y final = {y2:.3f} m")
```

Figura 16. Código 2.3: Coordenada final.

2.4.3 Problema 3. Área de un triángulo

Dado un triángulo con una base de 25.40 m y una altura de 12.35 m, calcular su área.

Solución:

$$A = 1/2 \times \text{base} \times \text{altura}$$

```python
base = 25.40
altura = 12.35

area = 0.5 * base * altura

print(f"Área del triángulo: {area:.2f} m^2")
```

Figura 17. Código 2.4: Área de un triángulo.

2.4.4 Problema 4. Clasificación didáctica de una pendiente

Dada una pendiente de 0.124, clasificarla utilizando, únicamente con fines didácticos, los siguientes intervalos:

- **Suave:** pendiente menor que 0.08.
- **Media:** pendiente entre 0.08 y 0.15.
- **Fuerte:** pendiente mayor que 0.15.

Estos intervalos se emplean exclusivamente para ilustrar el uso de estructuras condicionales en Python y no deben interpretarse como una clasificación normativa de pendientes.

Solución:

```
        pendiente = 0.124

        if pendiente < 0.08:
        clasificacion = "Suave"
        elif pendiente <= 0.15:
        clasificacion = "Media"
        else:
        clasificacion = "Fuerte"

        print(
        f"Pendiente: {pendiente:.3f} -> "
        f"Clasificación: {clasificacion}"
        )
```

Figura 18. Código 2.5: Clasificación didáctica de una pendiente.

2.4.5 Problema 5. Distancia entre dos puntos

Calcular la distancia horizontal entre $A(30.25, 50.40)$ y $B(80.90, 90.35)$.

Solución:

Se aplica la fórmula de la distancia euclidiana:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```python
import math

x1, y1 = 30.25, 50.40
x2, y2 = 80.90, 90.35

d = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

print(f"Distancia: {d:.2f} m")
```

Figura 19. Código 2.6: Distancia euclidiana.

2.4.6 Problema 6. Error de cierre lineal

En una poligonal cerrada, después del cálculo de los incrementos de coordenadas, se obtuvo un cierre de $fx = 0.18$ m y $fy = -0.24$ m. Determinar el error de cierre lineal.

Solución:

$$e = \sqrt{f_x^2 + f_y^2}$$

```python
import math

fx = 0.18
fy = -0.24

error_cierre = math.sqrt(fx**2 + fy**2)

print(f"Error de cierre lineal: {error_cierre:.3f} m")
```

Figura 20. Código 2.7: Error de cierre lineal.

2.5 Resumen

En este capítulo se presentaron los fundamentos de la planimetría y su aplicación al cálculo de coordenadas horizontales. Se revisaron los sistemas de coordenadas, los principales métodos de levantamiento planimétrico y el cálculo de incrementos a partir de azimutes y distancias. También se analizaron los errores de cierre angular y lineal en poligonales cerradas y la necesidad de aplicar procedimientos de ajuste para obtener coordenadas coherentes con la geometría del levantamiento.

Mediante los ejemplos desarrollados con Python se mostró cómo automatizar conversiones angulares, cálculos de coordenadas, distancias y errores de cierre. Estas herramientas permiten comprobar procedimientos y organizar el procesamiento de datos, manteniendo como elemento fundamental la interpretación técnica de las observaciones y de los resultados obtenidos.

2.6 Problemas propuestos

1. Cálculo de coordenadas. Desarrolle un programa que reciba las coordenadas iniciales (X, Y), un azimut expresado en grados y una distancia horizontal, y calcule las coordenadas del punto final.

2. Serie de puntos. Genere una lista de cinco azimutes y cinco distancias a partir de un punto inicial y calcule las coordenadas sucesivas de todos los puntos de la poligonal.

3. Verificación del cierre. Calcule las diferencias en las coordenadas X y Y entre el punto inicial y el punto final de una poligonal

cerrada. A partir de estas componentes, determine el error de cierre lineal.

4. Ajuste angular. Considere una poligonal cerrada de cinco vértices con ángulos interiores observados. Calcule la suma angular teórica mediante $(n-2) \times 180^{\circ}$, determine el error angular de cierre y distribuya la corrección entre los ángulos.

5. Conversión de rumbos a azimutes. Escriba una función que convierta rumbos cuadrantales, como $N45^{\circ}E$, $S30^{\circ}W$ y $N12^{\circ}W$, en azimutes expresados en grados decimales.

6. Aplicación práctica. Desde un punto inicial $A(1000.00, 1000.00)$ se observan los siguientes tramos: 100.40 m con azimut 45.75°, 80.25 m con azimut 120.30° y 90.10 m con azimut 210.45°. Calcule las coordenadas de cada vértice y determine la posición final del recorrido.

CAPÍTULO III

3 ALTIMETRÍA Y MODELO DIGITAL DEL TERRENO

La altimetría es la rama de la topografía encargada de determinar y representar las elevaciones de los puntos del terreno con respecto a una superficie o datum vertical de referencia. A diferencia de la planimetría, que describe la posición horizontal de los puntos, la altimetría incorpora la componente vertical y permite analizar el relieve, estudiar pendientes, elaborar perfiles y apoyar el diseño y control de diferentes obras de ingeniería (Wolf & Ghilani, 2012).

En ingeniería, la información altimétrica resulta fundamental para la elaboración de perfiles longitudinales y transversales, el diseño de carreteras y canales, el análisis del drenaje superficial, la evaluación de pendientes y la construcción de modelos digitales del terreno. Las herramientas computacionales y lenguajes como Python facilitan el procesamiento numérico y la representación gráfica de estos datos, permitiendo realizar interpolaciones, generar curvas de nivel y analizar superficies del terreno.

3.1 Cotas y diferencias de nivel

Una cota representa la elevación de un punto con respecto a una superficie o datum vertical de referencia. Si dos puntos A y B tienen cotas ZA y ZB, respectivamente, y se adopta como convención positiva el sentido de A hacia B, la diferencia de nivel se expresa como:

$$\Delta hAB = ZB - ZA$$

De acuerdo con esta convención, si $\Delta hAB > 0$, el punto B se encuentra a una elevación mayor que el punto A; si $\Delta hAB < 0$, el punto B se encuentra a una elevación menor. Si $\Delta hAB = 0$, ambos puntos presentan la misma cota.

La diferencia de nivel constituye un parámetro fundamental para el cálculo de pendientes, la elaboración de perfiles topográficos y el análisis de variaciones altimétricas del terreno.

Las cotas pueden determinarse mediante diferentes procedimientos, como la nivelación geométrica y la nivelación trigonométrica. También pueden obtenerse alturas mediante técnicas GNSS; sin embargo, debe considerarse el tipo de altura proporcionada y, cuando corresponda, la transformación necesaria para expresarla respecto del datum vertical utilizado en el proyecto.

3.2 Curvas de nivel

Las curvas de nivel son líneas que unen puntos que presentan una misma elevación con respecto a una superficie de referencia. Constituyen una forma convencional de representar el relieve del terreno en cartografía topográfica y permiten interpretar características como lomas, depresiones, valles, divisorias y cambios de pendiente.

El espaciamiento entre las curvas de nivel proporciona información sobre la variación del relieve:

- Las curvas muy próximas entre sí representan zonas de mayor pendiente.

- Las curvas más separadas representan zonas de menor pendiente.
- Las curvas cerradas pueden representar elevaciones o depresiones, cuya interpretación requiere considerar los valores de las cotas y la configuración del relieve.

Para generar curvas de nivel a partir de puntos dispersos puede construirse previamente una superficie mediante interpolación. Python permite implementar este procedimiento mediante bibliotecas como numpy, matplotlib y scipy, que proporcionan herramientas para el tratamiento numérico, la interpolación y la representación gráfica de datos (Virtanen et al., 2020).

Debe considerarse que la superficie resultante depende de la distribución y calidad de los puntos de entrada, así como del método de interpolación utilizado. Por tanto, las curvas obtenidas representan una estimación de la superficie a partir de los datos disponibles.

```
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy.interpolate import griddata

        Puntos medidos (X, Y, Z)

        puntos = np.array([
        [0.0, 0.0, 100.20],
        [0.0, 10.0, 102.10],
        [10.0, 0.0, 98.45],
        [10.0, 10.0, 101.35],
        [5.0, 5.0, 103.20]
        ])

        x = puntos[:, 0]
        y = puntos[:, 1]
        z = puntos[:, 2]

        Crear malla regular

        xg = np.linspace(0.0, 10.0, 100)
        yg = np.linspace(0.0, 10.0, 100)
        xi, yi = np.meshgrid(xg, yg)

        Interpolación lineal

        zi = griddata((x, y), z, (xi, yi), method="linear")

        Generar curvas de nivel

        curvas = plt.contour(xi, yi, zi, levels=12, cmap="terrain")
        plt.clabel(curvas, inline=True, fontsize=8)
        plt.scatter(x, y, c="black", s=20, label="Puntos medidos")
        plt.colorbar(curvas, label="Cota [m]")
        plt.title("Curvas de nivel interpoladas")
        plt.xlabel("X [m]")
        plt.ylabel("Y [m]")
        plt.legend()
        plt.grid(True)
        plt.axis("equal")
        plt.show()
```

Figura 21. Código3.1:Interpolaciónycurvasdenivel.

3.3 Modelo digital del terreno (MDT)

Un modelo digital del terreno (MDT) es una representación numérica del relieve en la que a cada posición del plano (X, Y) se le asigna un valor de elevación Z. Este tipo de modelo constituye la base para diversos análisis topográficos, hidrológicos y geoespaciales, como el

cálculo de volúmenes, la delimitación de cuencas, el análisis de pendientes, la generación de perfiles y la visualización tridimensional del relieve (Li, Zhu, & Gold, 2005).

En la práctica, conviene distinguir entre varios conceptos relacionados:

- **DEM (Digital Elevation Model):** modelo digital de elevación utilizado en sentido amplio para representar valores de altura o elevación.

- **DSM (Digital Surface Model):** representa la superficie incluyendo elementos ubicados sobre el terreno, como vegetación y edificaciones.

- **DTM o MDT:** representa de forma específica la superficie del terreno y puede incorporar procesos de filtrado y elementos relacionados con la geometría del relieve.

En diferentes fuentes técnicas, los términos DEM y DTM pueden emplearse con significados próximos o incluso intercambiables. Por ello, en aplicaciones de ingeniería es conveniente precisar el tipo de modelo utilizado y las características de los datos que lo componen.

3.3.1 Construcción del MDT a partir de puntos

Un MDT puede construirse a partir de un conjunto de puntos dispersos con coordenadas (X, Y, Z). Para ello, se define una malla regular y se estima la elevación correspondiente a cada nodo mediante un método de interpolación.

La calidad del modelo resultante depende de factores como la densidad y distribución espacial de los puntos, la precisión de las observaciones, el método de interpolación utilizado y la complejidad real del relieve.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import griddata

# Puntos con coordenadas (X, Y, Z)

puntos = np.array([
[0.0, 0.0, 102.30],
[0.0, 10.0, 104.15],
[10.0, 0.0, 100.40],
[10.0, 10.0, 106.20],
[5.0, 5.0, 110.10]
])

x = puntos[:, 0]
y = puntos[:, 1]
z = puntos[:, 2]

# Malla regular

xg = np.linspace(0.0, 10.0, 100)
yg = np.linspace(0.0, 10.0, 100)
xi, yi = np.meshgrid(xg, yg)

# Interpolación lineal

zi = griddata((x, y), z, (xi, yi), method="linear")

# Visualización en planta

plt.imshow(
zi,
extent=(0, 10, 0, 10),
origin="lower",
cmap="terrain"
)

plt.scatter(x, y, c="black", s=20)
plt.colorbar(label="Cota [m]")
plt.title("Modelo digital del terreno (MDT)")
plt.xlabel("X [m]")
plt.ylabel("Y [m]")
plt.grid(False)
plt.show()
```

Figura 22. Código3.2:GeneracióndeMDTinterpolado.

3.3.2 Visualización tridimensional del MDT

Además de la representación en planta, un MDT puede visualizarse en tres dimensiones para facilitar la interpretación de la geometría del relieve.

```
        import matplotlib.pyplot as plt

        fig = plt.figure()
        ax = fig.add_subplot(111, projection="3d")

        ax.plot_surface(
        xi,
        yi,
        zi,
        cmap="terrain",
        edgecolor="none"
        )

        ax.set_title("Modelo digital del terreno (3D)")
        ax.set_xlabel("X [m]")
        ax.set_ylabel("Y [m]")
        ax.set_zlabel("Cota [m]")

        plt.show()
```

Figura 23. Código 3.3: Visualización 3D del MDT.

3.3.3 Análisis de pendientes

El análisis de pendientes consiste en estimar la variación de la elevación en las direcciones principales de la malla. A partir del MDT, puede calcularse el gradiente del terreno y, con ello, la magnitud de la pendiente.

Si Δx y Δy representan la resolución espacial de la malla, una estimación de la pendiente es:

$$S = \sqrt{\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2}$$

donde $\partial z/\partial x$ y $\partial z/\partial y$ representan las variaciones de la elevación en las direcciones respectivamente.

El resultado S corresponde a una pendiente adimensional, que puede expresarse en porcentaje mediante:

$$S\,\% = S \times 100$$

También puede expresarse como ángulo de pendiente mediante:

$$\theta = \arctan(S)$$

```
dx = xg[1] - xg[0]
dy = yg[1] - yg[0]
grad_y, grad_x = np.gradient(zi, dy, dx)
pendiente = np.sqrt(grad_x**2 + grad_y**2)
plt.imshow(pendiente, extent=(0, 10, 0, 10),
origin='lower', cmap='plasma')
plt.colorbar(label='Pendiente [m/m]')
plt.title('Mapa de pendientes del MDT')
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.show()
```

Figura 24. Código 3.4: Análisis de pendientes.

3.4 Problemas resueltos

3.4.1 Problema 1. Diferencia de nivel entre dos puntos

Dado un punto A con cota 150.25 m y un punto B con cota 143.80 m, determinar la diferencia de nivel desde A hacia B.

Solución:

$$\Delta hAB = ZB - ZA$$

$$\Delta hAB = 143.80 - 150.25 = -6.45 \text{ m}$$

El signo negativo indica que el punto B se encuentra 6.45 m por debajo del punto A.

```
zA = 150.25
zB = 143.80
delta_h = zB - zA
print(f"Diferencia de nivel: {delta_h:.2f} m")
```

Figura 25. Código 3.5: Diferencia de nivel.

3.4.2 Problema 2. Perfil topográfico

Representar el perfil topográfico a lo largo de una línea de observación a partir de un conjunto de distancias y elevaciones.

Solución:

```
import numpy as np
import matplotlib.pyplot as plt

distancia = np.linspace(0.0, 108.0, 10)
elevacion = [
200.20, 202.10, 205.40, 210.25, 215.35,
212.40, 208.30, 205.15, 202.60, 200.45
]

plt.plot(distancia, elevacion, marker="o")
plt.xlabel("Distancia [m]")
plt.ylabel("Elevacion [m]")
plt.title("Perfil topografico")
plt.grid(True)
plt.show()
```

Figura 26. Código 3.6: Perfil topográfico.

3.4.3 Problema 3. Cálculo de pendiente entre dos puntos

Calcular la pendiente entre los puntos $A(0,\ 0,\ 150.20)$ y $B(100.00,\ 0,\ 175.45)$.

Solución:

La distancia horizontal entre los puntos se calcula mediante:

$$d_h = \sqrt{(X_B - X_A)^2 + (Y_B - Y_A)^2}$$

Para los datos del problema:

$$d_h = \sqrt{(100 - 0)^2 + (0 - 0)^2} = 100.00\,\text{m}$$

La diferencia de elevación es:

$$\Delta Z = ZB - ZA = 175.45 - 150.20 = 25.25\ \text{m}$$

Por tanto, la pendiente es:

$$S = \Delta Z / dh = 25.25/100.00 = 0.2525$$

En porcentaje:

$$S\ \% = 0.2525 \times 100 = 25.25\ \%$$

```python
import math

x1, y1, z1 = 0.0, 0.0, 150.20
x2, y2, z2 = 100.00, 0.0, 175.45

dh = math.sqrt((x2 - x1)**2 + (y2 - y1)**2)
pendiente = (z2 - z1) / dh
pendiente_porcentaje = pendiente * 100.0

print(f"Pendiente: {pendiente:.4f} m/m")
print(f"Pendiente: {pendiente_porcentaje:.2f} %")
```

Figura 27. Código 3.7: Pendiente entre dos puntos.

3.4.4 Problema 4. Generación de un MDT a partir de puntos

Construir un MDT a partir de un conjunto de puntos dispersos simulados mediante interpolación lineal.

Solución:

```python
from scipy.interpolate import griddata
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(25)
x = np.random.rand(80) * 100.0
y = np.random.rand(80) * 100.0
z = np.random.rand(80) * 45.0 + 150.0

xg = np.linspace(0.0, 100.0, 100)
yg = np.linspace(0.0, 100.0, 100)
xi, yi = np.meshgrid(xg, yg)
zi = griddata((x, y), z, (xi, yi), method="linear")

plt.imshow(
zi,
extent=(0, 100, 0, 100),
origin="lower",
cmap="terrain"
)
plt.colorbar(label="Elevacion [m]")
plt.title("MDT generado")
plt.xlabel("X [m]")
plt.ylabel("Y [m]")
plt.show()
```

Figura 28. Código3.8:Generación de un MDT.

3.4.5 Problema 5.Curvas de nivel de un MDT

Trazar curvas de nivel a partir del MDT interpolado en el problema anterior.

Solución:

```
        curvas = plt.contour(
        xi, yi, zi,
        levels=15,
        cmap="terrain"
        )
        plt.clabel(curvas, inline=True, fontsize=8)
        plt.title("Curvas de nivel del MDT")
        plt.xlabel("X [m]")
        plt.ylabel("Y [m]")
        plt.grid(True)
        plt.axis("equal")
        plt.show()
```

Figura 29. Código3.9:Curvas de nivel a partir de un MDT.

3.4.6 Problema 6. Cálculo de pendientes en una malla

Obtener un mapa de pendientes a partir del MDT interpolado.

Solución:

```
        dx = xg[1] - xg[0]
        dy = yg[1] - yg[0]

        grad_y, grad_x = np.gradient(zi, dy, dx)
        pendiente = np.sqrt(grad_x**2 + grad_y**2)

        plt.imshow(
        pendiente,
        extent=(0, 100, 0, 100),
        origin="lower",
        cmap="plasma"
        )
        plt.title("Mapa de pendientes")
        plt.colorbar(label="Pendiente [m/m]")
        plt.xlabel("X [m]")
        plt.ylabel("Y [m]")
        plt.show()
```

Figura 30. Código 3.10: Mapa de pendiente.

3.4.7 Problema 7. Clasificación didáctica del relieve

Clasificar el relieve en tres categorías según la magnitud de la pendiente calculada en el problema anterior. Para este ejercicio se adoptan, únicamente con fines didácticos, los siguientes intervalos: pendiente menor que 5 %, categoría suave; pendiente entre 5 % y 15 %, categoría media; y pendiente igual o superior al 15 %, categoría fuerte. Estos valores no corresponden a una clasificación normativa y pueden variar según la finalidad del estudio y la metodología empleada.

Solución:

```
        clasificacion = np.where(
        pendiente < 0.05,
        1,
        np.where(pendiente < 0.15, 2, 3)
        )

        plt.imshow(
        clasificacion,
        extent=(0, 100, 0, 100),
        origin="lower",
        cmap="Accent"
        )
        plt.title("Clasificacion didactica del relieve")
        plt.colorbar(label="Categoria")
        plt.xlabel("X [m]")
        plt.ylabel("Y [m]")
        plt.show()
```

Figura 31. Código 3.11: Clasificación didáctica del relieve.

3.4.8 Problema 8. Visualización de puntos sobre el MDT

Superponer los puntos originales sobre el MDT interpolado para verificar su distribución espacial respecto de la superficie generada.

Solución:

```
        plt.imshow(
        zi,
        extent=(0, 100, 0, 100),
        origin="lower",
        cmap="terrain"
        )
        plt.scatter(
        x, y,
        c=z,
        edgecolor="black",
        s=25,
        label="Puntos"
        )
        plt.legend()
        plt.title("Puntos sobre el MDT")
        plt.xlabel("X [m]")
        plt.ylabel("Y [m]")
        plt.colorbar(label="Elevacion [m]")
        plt.show()
```

Figura 32. Código 3.12:Puntos sobre el MDT.

3.4.9 Problema 9. Exportación del MDT

Exportar las coordenadas y elevaciones del MDT interpolado a un archivo CSV.

Solución:

```
        import pandas as pd

        df = pd.DataFrame({
                "x": xi.flatten(),
                "y": yi.flatten(),
                "z": zi.flatten()
        })

        df.to_csv("mdt.csv", index=False)
        print("Archivo MDT exportado")
```

Figura 33. Código 3.13: Exportación del MDT.

3.4.10 Problema 10. Estadísticas básicas del MDT

Obtener la media, el valor máximo y el valor mínimo de las elevaciones del MDT.

Solución:

```
        media = np.nanmean(zi)
        maximo = np.nanmax(zi)
        minimo = np.nanmin(zi)

        print(f"Media: {media:.2f} m")
        print(f"Maximo: {maximo:.2f} m")
        print(f"Minimo: {minimo:.2f} m")
```

Figura 34. Código 3.14: Estadísticas básicas del MDT.

3.5 Resumen

La altimetría estudia y representa las variaciones verticales del terreno mediante cotas, diferencias de nivel, perfiles y pendientes. La correcta interpretación del signo de la diferencia de nivel permite determinar la posición relativa entre puntos, mientras que el cálculo de pendientes facilita la caracterización de los cambios de elevación respecto de la distancia horizontal.

Las curvas de nivel constituyen una representación convencional del relieve y permiten reconocer visualmente su configuración. Por su parte, los modelos digitales del terreno permiten representar numéricamente una superficie y realizar análisis derivados, como la generación de curvas de nivel, perfiles topográficos, mapas de pendientes y estadísticas de elevación.

La construcción de un MDT a partir de puntos dispersos requiere considerar la densidad y distribución de las observaciones, su calidad y el método de interpolación empleado. En consecuencia, la superficie obtenida constituye una estimación del relieve basada en los datos disponibles.

Python proporciona herramientas para organizar, interpolar, analizar y visualizar información altimétrica mediante bibliotecas como NumPy, SciPy, Matplotlib y pandas. Estas herramientas permiten integrar los fundamentos topográficos con procedimientos computacionales reproducibles.

3.6 Problemas propuestos

1. Cálculo de diferencia de nivel: dado un punto A con cota 150.20 m y un punto B con cota 143.80 m, calcule la diferencia de nivel desde A hacia B y determine si existe ascenso o descenso.

2. Pendiente entre puntos: escriba un programa que reciba dos puntos con coordenadas (X, Y, Z) y calcule la distancia horizontal, la pendiente en porcentaje y el ángulo de pendiente en grados.

3. Interpolación lineal: implemente un programa en Python que determine la cota de un punto intermedio entre dos puntos conocidos mediante interpolación lineal.

4. Interpolación y visualización: a partir de seis puntos con coordenadas (X, Y, Z) definidas por el usuario, genere un MDT mediante interpolación y represente sus curvas de nivel.

5. Perfil longitudinal: dado un conjunto de puntos distribuidos a lo largo de una línea, represente gráficamente el perfil de elevaciones utilizando matplotlib.

6. Caso práctico: simule un levantamiento de 25 puntos distribuidos sobre una superficie de 50 × 50 m y genere su representación altimétrica mediante curvas de nivel.

7. Análisis de pendientes: escriba una función que reciba una matriz de elevaciones y devuelva una matriz con la magnitud aproximada de la pendiente en cada celda.

8. Interpolación IDW: investigue e implemente el método Inverse Distance Weighting (IDW) para generar un MDT a partir de puntos dispersos.

9. Visualización del terreno: simule un terreno ondulado mediante un conjunto de puntos y visualice el MDT resultante en dos y tres dimensiones. Interprete las principales características del relieve.

10. Mapa de pendientes: a partir de un MDT simulado de 5×5 celdas, calcule e interprete el mapa de pendientes e identifique las zonas de mayor inclinación.

11. Mosaico de modelos: genere dos MDT correspondientes a sectores adyacentes, intégrelos en un modelo continuo y represente gráficamente el resultado.

12. Comparación de interpolaciones: genere un MDT mediante interpolación lineal, otro mediante interpolación cúbica y otro mediante IDW. Compare visualmente los resultados y discuta las principales diferencias.

13. Generación automática de curvas: utilice un MDT para generar curvas de nivel con una equidistancia vertical de 0.50 m. Incorpore etiquetas e información gráfica adecuada.

14. Estudio de resolución: genere MDT con resoluciones espaciales de 1 m, 5 m y 10 m. Compare gráficamente el efecto de la resolución sobre la representación del relieve.

CAPÍTULO IV

4 CÁLCULO DE VOLÚMENES EN TOPOGRAFÍA APLICADA

El cálculo de volúmenes es una de las aplicaciones más importantes de la topografía en obras civiles, movimientos de tierra, plataformas, canales, presas, carreteras y proyectos de urbanización. A partir de la geometría del terreno natural y de la superficie proyectada, es posible estimar cantidades de corte y relleno, evaluar costos, programar maquinaria y controlar el avance de obra. Por ello, la determinación confiable de volúmenes constituye una tarea esencial dentro de la ingeniería topográfica y de la ingeniería civil en general (Wolf & Ghilani, 2012).

Existen diversos métodos para estimar volúmenes, desde procedimientos geométricos clásicos basados en secciones hasta técnicas digitales apoyadas en modelos digitales del terreno (MDT). La elección del método depende de la forma del terreno, de la densidad de la información disponible, de la precisión requerida y del tipo de proyecto. En la actualidad, Python permite automatizar estos procedimientos, reducir errores de cálculo y generar representaciones visuales que facilitan la interpretación técnica de los resultados.

4.1 Fundamento geométrico y matemático

Desde el punto de vista matemático, el volumen de una masa de terreno puede interpretarse como la integral de una superficie de alturas sobre una región del plano. En topografía aplicada, este cálculo continuo suele

aproximarse mediante procedimientos discretos, porque en la práctica se trabaja con puntos medidos, perfiles, secciones o rejillas regulares.

Si $z(x, y)$ representa la diferencia de elevación entre una superficie real y una superficie de referencia, entonces el volumen puede expresarse, de forma ideal, como:

$$V = \iint_A z(x, y)\, dA$$

donde A es el área en planta sobre la cual se realiza la integración. Como normalmente no se dispone de una función continua exacta, el volumen se aproxima mediante métodos numéricos apropiados al tipo de levantamiento.

4.2 Método de secciones transversales

Este método se emplea ampliamente en obras lineales, como carreteras, canales, ferrocarriles y tuberías. Consiste en determinar las áreas de secciones transversales sucesivas a lo largo de un eje longitudinal y estimar el volumen comprendido entre dos secciones consecutivas.

La forma más simple corresponde al método del promedio de áreas extremas:

$$V = \frac{d}{2}(A_1 + A_2)$$

donde:

d es la distancia entre las secciones;

$A1$ y $A2$ son las áreas de las secciones extremas.

Este método es práctico y de aplicación directa, aunque supone que la variación del área entre ambas secciones puede aproximarse de forma lineal.

4.3 Método prismoidal

El método prismoidal puede proporcionar una mejor aproximación del volumen cuando se dispone del área de una sección intermedia y la variación entre las secciones extremas no puede considerarse estrictamente lineal. Su expresión es:

$$V = \frac{L}{6}(A_1 + 4A_m + A_2)$$

donde:

L es la distancia entre las secciones extremas;

$A1$ es el área de la sección inicial;

Am es el área de la sección media;

$A2$ es el área de la sección final.

Esta expresión se basa en una aproximación parabólica de la variación de las áreas y puede ofrecer resultados más representativos que el método del promedio de áreas extremas cuando la geometría entre secciones presenta variaciones suaves y no lineales.

4.4 Método de cuadrícula

En este método, el terreno se divide en celdas regulares de área conocida. Para cada celda se determina una altura media o una diferencia de elevación respecto de una superficie de referencia y, posteriormente, el volumen total se obtiene mediante la suma de los volúmenes parciales.

Si hi representa la altura media correspondiente a la celda i y Acelda el área de cada celda, entonces:

$$V = \sum_{i=1}^{n} h_i \, A_{\text{celda}}$$

En aplicaciones de movimiento de tierras, el signo de la diferencia de elevación debe interpretarse de acuerdo con la convención adoptada. Cuando se utiliza la diferencia entre la superficie natural y la superficie de proyecto, un valor positivo indica que el terreno natural se encuentra por encima de la superficie proyectada y, por tanto, corresponde a corte; un valor negativo corresponde a relleno.

4.5 Volúmenes por diferencia de superficies

Cuando se dispone de dos superficies digitales, por ejemplo, el terreno natural y la superficie de proyecto, el volumen puede estimarse a partir de la diferencia de elevación entre ambos modelos. Si Znatural y Zproyecto representan las elevaciones de las dos superficies, se adopta:

$$\Delta Z = Z\text{natural} - Z\text{proyecto}$$

Con esta convención, los valores positivos de ΔZ corresponden a zonas de corte, mientras que los valores negativos corresponden a zonas de relleno.

Para una cuadrícula regular, el volumen algebraico puede aproximarse mediante:

$$V = \sum_{i=1}^{n} \Delta Z_i \, A_{\text{celda}}$$

En aplicaciones prácticas, resulta conveniente calcular por separado los volúmenes de corte y relleno, ya que el volumen algebraico total puede ocultar compensaciones entre ambos.

Este enfoque es especialmente útil en topografía computacional, porque permite analizar de forma sistemática las diferencias entre superficies digitales y representar espacialmente las zonas de corte y relleno.

4.6 Cálculo de volúmenes con Python

Python permite automatizar los procedimientos de cálculo de volúmenes y facilita la revisión, reproducción y visualización de los resultados. A continuación, se presentan ejemplos básicos de implementación.

4.6.1 Volumen por método prismoidal

```
1    def volumen_prismoide(A1, Am, A2, L):
2    return (L / 6.0) * (A1 + 4.0 * Am + A2)
3
4    v = volumen_prismoide(45.80, 50.35, 52.10, 20.50)
5    print(f"Volumen prismoidal: {v:.2f} m3")
```

Figura 35. Código 4.1: Volumen por método prismoidal.

4.6.2 *Volumen por promedio de áreas extremas*

```
        def volumen_secciones(A1, A2, d):
        return (d / 2.0) * (A1 + A2)

        v = volumen_secciones(38.45, 44.20, 18.75)
        print(f"Volumen entre secciones: {v:.2f} m3")
```

Figura 36. Código 4.2: Volumen por secciones transversales.

4.6.3 *Volumen por diferencia de MDT*

```
        import numpy as np

        def volumen_mdt(mdt_natural, mdt_proyecto, resolucion):
        delta = mdt_natural - mdt_proyecto
        area_celda = resolucion**2
        corte = np.where(delta > 0, delta, 0)
        relleno = np.where(delta < 0, -delta, 0)
        volumen_corte = np.nansum(corte) * area_celda
        volumen_relleno = np.nansum(relleno) * area_celda
        return volumen_corte, volumen_relleno, delta

        mdt_natural = np.array([
        [100.20, 101.10],
        [99.85, 100.75]
        ])

        mdt_proyecto = np.array([
        [98.90, 99.80],
        [98.10, 99.20]
        ])

        vc, vr, delta = volumen_mdt(
        mdt_natural,
        mdt_proyecto,
        10.0
        )

        print(f"Volumen de corte: {vc:.2f} m3")
        print(f"Volumen de relleno: {vr:.2f} m3")
```

Figura 37. Código 4.3: Volumen por diferencia de MDT.

4.6.4 Visualización de cortes y rellenos

```
import matplotlib.pyplot as plt

plt.imshow(delta, cmap="RdYlGn", origin="lower")
plt.colorbar(label="Diferencia de elevacion [m]")
plt.title("Mapa de corte y relleno")
plt.xlabel("Columnas")
plt.ylabel("Filas")
plt.axis("equal")
plt.show()
```

Figura 38. Código 4.4: Mapa de cortes y rellenos.

4.6.5 Histograma de diferencias de elevación

```
plt.hist(
delta.flatten(),
bins=6,
color="gray",
edgecolor="black"
)

plt.title("Distribucion de diferencias de elevacion")
plt.xlabel("Diferencia de cota [m]")
plt.ylabel("Numero de celdas")
plt.grid(True)
plt.show()
```

Figura 39. Código 4.5: Histograma de diferencias de elevación.

4.7 Comparación de métodos

Cada método presenta ventajas y limitaciones:

El método de secciones transversales es apropiado para obras lineales y resulta sencillo de aplicar, aunque sus resultados dependen, entre otros factores, de la separación entre secciones y de la variación real del terreno entre ellas.

- El método prismoidal incorpora información de una sección intermedia y puede representar mejor ciertas variaciones no lineales entre secciones.

- El método de cuadrícula es útil en explanaciones, plataformas y superficies extensas, especialmente cuando se dispone de información distribuida regularmente.

- La diferencia entre MDT facilita el análisis computacional de superficies completas y permite identificar espacialmente las zonas de corte y relleno.

La elección del método no depende únicamente de la fórmula utilizada, sino también de la calidad y densidad de los datos, de la geometría del terreno, de la resolución espacial de los modelos y de las exigencias del proyecto. En particular, la separación entre secciones puede influir de manera impor-tante en las estimaciones obtenidas mediante el método de áreas extremas, mientras que los métodos basados en superficies digitales dependen de la calidad de la modelación del terreno.

4.8 Problemas resueltos

4.8.1 Problema 1. Volumen entre dos secciones

Dos secciones transversales consecutivas tienen áreas de 40.25 m2 y 55.80 m2, separadas por una distancia de 25.40 m. Calcule el volumen entre ellas mediante el método del promedio de áreas extremas.

Solución:

$$V = \frac{d}{2}(A_1 + A_2)$$

$$V = \frac{25.40}{2}(40.25 + 55.80) = 1219.84\,\text{m}^3$$

```
A1 = 40.25
A2 = 55.80
d = 25.40
V = (d / 2.0) * (A1 + A2)
print(f"Volumen: {V:.2f} m3")
```

Figura 40. Código 4.6: Volumen entre dos secciones.

4.8.2 Problema 2. Volumen prismoidal

Se conocen las áreas $A1$ = 45.80 m2, Am = 50.35 m2 y $A2$ = 52.10 m2, con una distancia entre las secciones extremas de 20.50 m. Calcule el volumen mediante el método prismoidal.

Solución:

$$V = \frac{L}{6}(A_1 + 4A_m + A_2)$$

$$V = \frac{20.50}{6}(45.80 + 4(50.35) + 52.10) = 1022.61\,\text{m}^3$$

```
A1 = 45.80
Am = 50.35
A2 = 52.10
L = 20.50
V = (L / 6.0) * (A1 + 4.0 * Am + A2)
print(f"Volumen prismoidal: {V:.2f} m3")
```

Figura 41. Código 4.7: Volumen por método prismoidal.

4.8.3 Problema 3. Volumen por cuadrícula

Una superficie se divide en celdas de 5.0 m × 5.0 m. Las alturas medias de cuatro celdas respecto de un plano de referencia son 0.85, 1.10, 0.95 y

1.25 m. Determine el volumen total.

Solución:

El área de cada celda es:

$$A_{\text{celda}} = 5.0 \times 5.0 = 25.0\,\text{m}^2$$

volumen total se obtiene mediante:

$$V = \sum_{i=1}^{4} h_i A_{\text{celda}}$$

$$V = (0.85 + 1.10 + 0.95 + 1.25) \times 25.0 = 103.75\,\text{m}^3$$

```python
import numpy as np

alturas = np.array([0.85, 1.10, 0.95, 1.25])
area_celda = 25.0
V = np.sum(alturas) * area_celda

print(f"Volumen total: {V:.2f} m3")
```

Figura 42. Código 4.8: Volumen por cuadrícula.

4.8.4 Problema 4. Corte por diferencia entre MDT

Dados dos MDT de 2 × 2, determine el volumen de corte considerando una resolución de celda de 10.0 m.

Se adopta la convención:

$$\Delta Z = Z\text{natural} - Z\text{proyecto}$$

por lo que los valores positivos representan corte y los valores negativos representan relleno.

Solución:

```
import numpy as np

mdt_natural = np.array([
[102.30, 101.80],
[100.95, 100.40]
])

mdt_proyecto = np.array([
[101.00, 100.50],
[100.00, 99.60]
])

resolucion = 10.0
area_celda = resolucion**2

delta = mdt_natural - mdt_proyecto
volumen_corte = np.sum(delta[delta > 0]) * area_celda

print(f"Volumen de corte: {volumen_corte:.2f} m3")
print("Matriz de diferencias:")
print(delta)
```

Figura 43. Código 4.9: Diferencia entre MDT.

Para los datos del problema, todas las diferencias son positivas, por lo que el resultado corresponde únicamente a corte:

$$V_{\text{corte}} = 435.00\,\text{m}^3$$

4.8.5 Problema 5. Separación entre corte y relleno

A partir de una matriz de diferencias de elevación, determine por separado los volúmenes de corte y relleno.

Se mantiene la convención de que los valores positivos representan corte y los valores negativos representan relleno.

Solución:

```
import numpy as np

delta = np.array([
[1.20, -0.80],
[0.55, -1.10]
])

resolucion = 8.0
area_celda = resolucion**2

corte = np.sum(delta[delta > 0]) * area_celda
relleno = -np.sum(delta[delta < 0]) * area_celda

print(f"Volumen de corte: {corte:.2f} m3")
print(f"Volumen de relleno: {relleno:.2f} m3")
```

Figura 44. Código 4.10: Separación de corte y relleno.

Los resultados son: $V_{\text{corte}} = 112.00\,\text{m}^3$ y $V_{\text{relleno}} = 121.60\,\text{m}^3$.

4.8.6 Problema 6. Histograma de diferencias

Representar mediante un histograma la distribución de las diferencias de elevación entre dos superficies.

Solución:

```
import numpy as np
import matplotlib.pyplot as plt

delta = np.array([
[1.20, -0.80, 0.35],
[0.55, -1.10, 0.10],
[0.25, -0.40, 0.95]
])

plt.hist(delta.flatten(), bins=6,
color="gray", edgecolor="black")
plt.title("Distribucion de diferencias")
plt.xlabel("Diferencia de elevacion [m]")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()
```

Figura 45. Código4.11:Histogramadediferencias.

4.8.7 Problema 7. Volumen acumulado entre secciones

Calcular el volumen acumulado entre cuatro secciones sucesivas mediante el método del promedio de áreas extremas.

Solución:

```
areas = [35.40, 38.25, 41.10, 39.85]
distancias = [20.0, 20.0, 20.0]

volumen_total = 0.0

for i in range(len(distancias)):
V = (distancias[i] / 2.0) * (
areas[i] + areas[i + 1]
)
volumen_total += V

print(f"Volumen acumulado: {volumen_total:.2f} m3")
```

Figura 46. Código4.12:Volumenacumuladoporsecciones.

El volumen acumulado es:

$$V_{\text{total}} = 2339.50\,\text{m}^3$$

4.8.8 Problema 8. Mapa de diferencias de elevación

Representar gráficamente una matriz de diferencias de elevación para identificar espacialmente zonas de corte y relleno.

Solución:

```
import numpy as np
import matplotlib.pyplot as plt

delta = np.array([
[0.8, 1.1, 0.6],
[-0.2, 0.4, -0.5],
[-1.0, -0.3, 0.2]
])

plt.imshow(delta, cmap="RdYlGn", origin="lower")
plt.colorbar(label="Diferencia de elevacion [m]")
plt.title("Mapa de corte y relleno")
plt.xlabel("Columnas")
plt.ylabel("Filas")
plt.show()
```

Figura 47. Código 4.13: Mapa de diferencias de elevación.

4.8.9 Problema 9. Cálculo del volumen de una rampa

Modelar una superficie en forma de rampa sobre un área de 20 m × 20 m y calcular su volumen aproximado respecto de un plano horizontal de referencia.

Para evitar considerar los nodos de la malla como si fueran celdas completas, se calcula la elevación media de cada celda a partir de sus cuatro vértices.

Solución:

```
import numpy as np

x = np.linspace(0.0, 20.0, 5)
y = np.linspace(0.0, 20.0, 5)
xi, yi = np.meshgrid(x, y)

z = 0.05 * xi + 0.02 * yi
dx = x[1] - x[0]
dy = y[1] - y[0]
area_celda = dx * dy

z_media = (
z[:-1, :-1] + z[1:, :-1] +
z[:-1, 1:] + z[1:, 1:]
) / 4.0

V = np.sum(z_media) * area_celda
print(f"Volumen aproximado: {V:.2f} m3")
```

Figura 48.Código4.14:Volumen de una rampa.

4.8.10 Problema 10. Estadísticas básicas de diferencias

Obtener la media, el valor máximo y el valor mínimo de una matriz de diferencias altimétricas.

Solución:

```
import numpy as np

delta = np.array([
[1.20, -0.80, 0.35],
[0.55, -1.10, 0.10],
[0.25, -0.40, 0.95]
])

print(f"Media: {np.mean(delta):.3f} m")
print(f"Maximo: {np.max(delta):.3f} m")
print(f"Minimo: {np.min(delta):.3f} m")
```

Figura 49. Código4.15:Estadísticas de diferencias.

4.9 Resumen

En este capítulo se estudiaron métodos geométricos y computacionales para la estimación de volúmenes en topografía aplicada. El método del promedio de áreas extremas permite calcular volúmenes entre secciones consecutivas, mientras que el método prismoidal incorpora una sección intermedia para representar de mejor manera ciertas variaciones de la geometría entre secciones.

El método de cuadrícula permite aproximar volúmenes mediante la suma de contribuciones parciales asociadas a celdas de área conocida. Cuando se dispone de dos superficies digitales, la diferencia entre sus elevaciones permite identificar y cuantificar de manera independiente las zonas de corte y relleno.

La confiabilidad de una estimación volumétrica no depende únicamente del procedimiento matemático empleado. También intervienen la calidad de los datos topográficos, la densidad de las observaciones, la separación entre secciones, la resolución espacial y la capacidad del modelo para representar las variaciones reales del terreno.

Python permite automatizar estos procedimientos mediante estructuras numéricas y herramientas de visualización, facilitando el cálculo de volúmenes, el análisis de diferencias altimétricas y la representación espacial de los movimientos de tierra.

4.10 Problemas propuestos

1. Calcule el volumen entre dos secciones con áreas de 40.50 m2 y

55.25 m2, separadas por una distancia de 25.80 m, mediante el método del promedio de áreas extremas.

2. Simule un MDT de 5×5 celdas, compárelo con un plano horizontal de referencia y determine por separado los volúmenes de corte y relleno.

3. Cree una función en Python que reciba los valores $A1$, Am, $A2$ y L, y devuelva el volumen calculado mediante el método prismoidal.

4. Compare el volumen obtenido mediante el método del promedio de áreas extremas y el método prismoidal para un conjunto de tres secciones. Analice las diferencias entre los resultados.

5. Genere dos superficies mediante interpolación lineal y cúbica a partir del mismo conjunto de puntos. Compare los volúmenes obtenidos respecto de un plano de referencia.

6. Implemente un mapa de diferencias entre un terreno natural y una superficie de proyecto e identifique visualmente las zonas de corte y relleno.

7. Modele una superficie en forma de rampa y calcule su volumen res-pecto de un plano horizontal mediante una cuadrícula regular.

8. A partir de una matriz de diferencias de elevación, determine de forma independiente los volúmenes de corte y relleno.

9. Genere un histograma de diferencias altimétricas y explique qué información aporta al análisis del movimiento de tierras.

10. Construya una rutina que calcule el volumen acumulado entre varias secciones sucesivas de una obra lineal y muestre el volumen correspondiente a cada intervalo.

CAPÍTULO V

5 GNSS Y GEORREFERENCIACIÓN EN TOPOGRAFÍA CON PYTHON

Los sistemas GNSS (Global Navigation Satellite Systems) permiten obtener coordenadas georreferenciadas mediante señales transmitidas por constelaciones de satélites. En la topografía moderna, su incorporación ha mejorado los procesos de levantamiento, replanteo, control de obras y vinculación a marcos de referencia geodésicos, aportando rapidez, cobertura espacial y altos niveles de precisión cuando se emplean metodologías adecuadas (El-Rabbany, 2002; Seeber, 2003).

Desde el punto de vista de la ingeniería topográfica, el uso de GNSS no se limita a determinar la posición de un punto. También permite integrar levantamientos a sistemas de referencia geodésicos, establecer bases de control, georreferenciar cartografía, controlar desplazamientos y vincular información de campo con sistemas de información geográfica y modelos digitales del terreno. Python, por su parte, proporciona herramientas para procesar, transformar, visualizar y analizar este tipo de datos espaciales de manera reproducible.

5.1 Fundamentos de los sistemas GNSS

Entre los principales sistemas globales de navegación por satélite se encuentran:

- **GPS:** desarrollado por Estados Unidos;

- **GLONASS:** desarrollado por Rusia;
- **Galileo:** desarrollado por la Unión Europea;
- **BeiDou:** desarrollado por China.

Un receptor GNSS determina su posición a partir de observaciones realizadas a varios satélites. Una de las observaciones fundamentales es la pseudodistancia, que constituye una estimación de la distancia entre el satélite y el receptor afectada por diferentes fuentes de error.

Para determinar una posición tridimensional y estimar simultáneamente el sesgo del reloj del receptor se requieren, en términos generales, observaciones de al menos cuatro satélites. El principio geométrico se basa en la determinación espacial de la posición mediante distancias, aunque en la práctica intervienen también las efemérides satelitales, los errores de reloj, los efectos ionosféricos y troposféricos, la geometría de los satélites y otros factores asociados al entorno de observación.

La precisión obtenida depende del método de posicionamiento, del receptor, de la geometría satelital, de las condiciones de observación y del procesamiento aplicado. De forma general, pueden distinguirse los siguientes métodos:

- **Posicionamiento autónomo:** utiliza observaciones de un único receptor y generalmente proporciona precisiones del orden de metros;

- **Posicionamiento diferencial:** emplea correcciones procedentes de una estación de referencia y permite mejorar la precisión respecto del posicionamiento autónomo;

- **RTK (*Real-Time Kinematic*):** utiliza observaciones de fase de la portadora y correcciones transmitidas en tiempo real, pudiendo alcanzar precisión centimétrica bajo condiciones adecuadas de observación, geometría satelital y comunicación.

5.2 Sistemas de coordenadas y georreferenciación

En topografía aplicada con GNSS es indispensable distinguir entre varios conceptos relacionados con la representación espacial de las coordenadas:

- **Coordenadas geográficas:** expresan la posición mediante latitud y longitud referidas a un elipsoide y a un sistema de referencia geodésico;
- **Coordenadas proyectadas:** representan posiciones sobre un plano mediante una proyección cartográfica, como UTM;
- **Sistema de referencia geodésico:** establece el marco utilizado para definir y expresar las coordenadas de los puntos;
- **Georreferenciación:** proceso mediante el cual un levantamiento o conjunto de datos espaciales se vincula a un sistema de referencia conocido, permitiendo su integración con cartografía, SIG y otros productos geoespaciales.

5.2.1 Coordenadas geográficas

Las coordenadas geográficas están formadas principalmente por:

- **Latitud:** ángulo que determina la posición de un punto al norte o al sur del ecuador;

- **Longitud:** ángulo que determina la posición de un punto al este o al oeste del meridiano de referencia.

Estas coordenadas permiten expresar una posición sobre un elipsoide de referencia. Sin embargo, en numerosos trabajos topográficos resulta más práctico utilizar coordenadas proyectadas expresadas en metros, ya que facilitan operaciones como el cálculo de distancias, áreas y replanteos dentro de una zona de trabajo determinada.

5.2.2 Coordenadas UTM

El sistema UTM (Universal Transverse Mercator) divide convencionalmente la superficie terrestre en 60 zonas longitudinales de 6° de amplitud. Dentro de cada zona se utiliza una proyección transversa de Mercator y las coordenadas se expresan en metros mediante los componentes Este (E) y Norte (N).

En ingeniería topográfica, el uso correcto de coordenadas UTM requiere identificar la zona, el hemisferio y el sistema de referencia geodésico asociado. Utilizar únicamente los valores de Este y Norte sin esta información puede producir ambigüedades o errores en la ubicación de los datos.

5.3 Transformación entre sistemas de coordenadas

Python dispone de la biblioteca pyproj, que permite realizar transformaciones entre sistemas de referencia de coordenadas. Esta herramienta

resulta útil para convertir coordenadas geográficas en coordenadas proyectadas, o efectuar la transformación inversa, siempre que se definan correctamente los sistemas de referencia de origen y destino.

En el siguiente ejemplo se transforman coordenadas geográficas referidas a WGS 84, identificadas mediante el código EPSG:4326, a coordenadas UTM zona 17 Sur, correspondientes al código EPSG:32717.

```
from pyproj import Transformer

transformador = Transformer.from_crs(
"EPSG:4326",
"EPSG:32717",
always_xy=True
)

lon, lat = -79.1200, -3.9900
este, norte = transformador.transform(lon, lat)

print(f"Este: {este:.2f} m")
print(f"Norte: {norte:.2f} m")
```

Figura 50. Código 5.1: Conversión de WGS 84 a UTM zona 17 Sur.

El parámetro always_xy=True permite mantener el orden de entrada como longitud y latitud. En este caso, el resultado se expresa mediante coordenadas Este y Norte en metros dentro de la zona UTM 17 Sur.

Este tipo de transformación es fundamental cuando se requiere convertir observaciones GNSS expresadas en latitud y longitud a coordenadas planas utilizadas en replanteo, diseño y cálculo topográfico. No obstante, una transformación de coordenadas no debe interpretarse únicamente como un cambio de unidades, ya que es necesario considerar correctamente el sistema de referencia geodésico, la proyección

cartográfica y, cuando corresponda, las transformaciones entre diferentes marcos o datums.

5.4 Manejo de datos espaciales con GeoPandas

La biblioteca GeoPandas permite trabajar con datos espaciales estructurados como tablas georreferenciadas. Esto facilita la lectura, creación, transformación y visualización de puntos GNSS, trayectorias y capas vectoriales dentro de entornos de análisis espacial.

```
import geopandas as gpd
from shapely.geometry import Point

coordenadas = [
(-79.1200, -3.9900),
(-79.1300, -4.0100)
]

geometrias = [Point(xy) for xy in coordenadas]

gdf = gpd.GeoDataFrame(
{"id": [1, 2]},
geometry=geometrias,
crs="EPSG:4326"
)

print(gdf)
```

Figura 51. Código 5.2: Creación de un Geo Data Frame con coordenadas GNSS.

Una vez creado, el GeoDataFrame puede reproyectarse a un sistema UTM y utilizarse para medir distancias, generar mapas o integrarse con otros datos geográficos.

```
gdf_utm = gdf.to_crs("EPSG:32717")
print(gdf_utm)
```

Figura 52. Código 5.3: Transformación de GeoDataFrame a UTM.

En este ejemplo se utiliza el sistema UTM zona 17 Sur, correspondiente al código EPSG:32717. La reproyección permite expresar las coordenadas en metros, lo que facilita operaciones métricas dentro del área de estudio.

5.5 Visualización de trayectorias GNSS

En levantamientos dinámicos, monitoreo de rutas o control de desplazamientos, es habitual representar trayectorias GNSS como secuencias ordenadas de coordenadas. Una visualización básica puede realizarse directamente con matplotlib.

```
import matplotlib.pyplot as plt

longitudes = [
-79.1200, -79.1250,
-79.1320, -79.1400
]

latitudes = [
-3.9900, -3.9950,
-4.0020, -4.0080
]

plt.plot(longitudes, latitudes, marker="o")
plt.title("Trayectoria GNSS")
plt.xlabel("Longitud")
plt.ylabel("Latitud")
plt.grid(True)
plt.show()
```

Figura 53. Código 5.4: Visualización de trayectoria GNSS.

Cuando se requiere calcular distancias recorridas, áreas u otras magnitudes métricas, es conveniente transformar previamente las coordenadas geográficas a un sistema de coordenadas proyectadas adecuado. De esta manera, los cálculos se realizan en unidades lineales y no directamente sobre diferencias angulares expresadas en grados.

5.6 Errores en posicionamiento GNSS

El posicionamiento GNSS está afectado por diversas fuentes de error que pueden influir en la precisión de las coordenadas obtenidas. Entre las más relevantes se encuentran:

- **Multitrayectoria:** ocurre cuando una señal llega al receptor después de reflejarse en superficies cercanas, como edificaciones, estructuras metálicas o el terreno;

- **Efectos ionosféricos:** producen variaciones en la propagación de las señales GNSS al atravesar la ionosfera;

- **Efectos troposféricos:** generan retardos en la propagación de la señal debido a las condiciones de la atmósfera neutra;

- **Errores orbitales:** están relacionados con las incertidumbres exis-tentes en la posición calculada de los satélites;

- **Errores de reloj:** corresponden a desviaciones de los relojes de los satélites y del receptor respecto del tiempo de referencia;

- **Dilución de precisión (DOP):** expresa el efecto de la geometría relativa entre los satélites y el receptor sobre la propagación de los errores de las observaciones hacia la solución de posicionamiento.

Una geometría satelital favorable contribuye a obtener valores menores de DOP y, en igualdad de otras condiciones, una solución geométricamente más favorable. Sin embargo, el DOP no representa por sí solo la precisión final, ya que esta también depende de la calidad de

las observaciones y de otras fuentes de error presentes en el posicionamiento.

5.7 Problemas resueltos

5.7.1 Problema 1. Conversión de coordenadas geográficas a UTM

Convierta las coordenadas (latitud = −2.5000◦, longitud = −76.5000◦) a coordenadas UTM.

Solución:

La longitud corresponde a la zona UTM 18 y, debido a que la latitud es negativa, el punto se encuentra en el hemisferio sur. Por tanto, se utiliza UTM zona 18 Sur, correspondiente al código EPSG:32718.

```
from pyproj import Transformer

transformador = Transformer.from_crs(
"EPSG:4326",
"EPSG:32718",
always_xy=True
)

lon, lat = -76.5000, -2.5000
este, norte = transformador.transform(lon, lat)

print(f"Este: {este:.2f} m")
print(f"Norte: {norte:.2f} m")
```

Figura 54. Código 5.5: Coordenadas geográficas a UTM

5.7.2 Problema 2. Determinación de la zona UTM

Determine a qué zona UTM pertenece un punto con longitud −70.00◦.

Solución:

La zona UTM puede obtenerse mediante:

$$\text{zona} = \left\lfloor \frac{\lambda + 180^\circ}{6^\circ} \right\rfloor + 1$$

Para $\lambda = -70.00^\circ$, se obtiene la zona UTM 19.

```
import math

longitud = -70.00
zona = math.floor((longitud + 180.0) / 6.0) + 1

print(f"La zona UTM es: {zona}")
```

Figura 55. Código 5.6: Determinación de la zona UTM.

5.7.3 Problema 3. Tiempo de viaje de una señal GNSS

Calcule el tiempo aproximado que tarda una señal electromagnética en recorrer una distancia de 20 200 km, considerando la velocidad de la luz en el vacío.

Solución:

El tiempo se determina mediante:

$$t = \frac{d}{c}$$

donde d es la distancia recorrida y c es la velocidad de la luz.

$$t = \frac{20\,200\,000}{299\,792\,458} \approx 0.06738\ \text{s}$$

```
distancia = 20200e3
velocidad_luz = 299792458.0

tiempo = distancia / velocidad_luz

print(f"Tiempo de viaje: {tiempo:.6f} s")
```

Figura 56. Código 5.7: Tiempo de viaje de una señal GNSS.

5.7.4 Problema 4. Conversión de grados decimales a sexagesimales

Convierta 4.7110◦ a grados, minutos y segundos.

Solución:

El resultado de la conversión es: 4.7110◦ = 4◦ 42′ 39.60′′.

```
decimal = 4.7110

grados = int(decimal)
minutos_dec = (decimal - grados) * 60
minutos = int(minutos_dec)
segundos = (minutos_dec - minutos) * 60

print(
f"{decimal} grados = "
f"{grados} grados {minutos} min "
f"{segundos:.2f} s"
)
```

Figura 57. Código 5.8: Conversión de decimal a sexagesimal.

5.7.5 Problema 5. Determinación del hemisferio

Identifique si una latitud de −3.99◦ corresponde al hemisferio norte o sur.

Solución:

Una latitud negativa corresponde al hemisferio sur, mientras que una latitud positiva corresponde al hemisferio norte. Por tanto, el punto se encuentra en el hemisferio sur.

```
latitud = -3.99

hemisferio = "Norte" if latitud >= 0 else "Sur"

print(f"Hemisferio: {hemisferio}")
```

Figura 58. Código 5.9: Determinación del hemisferio.

5.7.6 Problema 6. Cálculo de PDOP y GDOP

Si $HDOP$ = 1.9, $VDOP$ = 1.2 y $TDOP$ = 1.1, calcule $PDOP$ y $GDOP$.

Solución:

Se emplean las relaciones:

$$PDOP = \sqrt{HDOP^2 + VDOP^2}$$

Y:

$$GDOP = \sqrt{PDOP^2 + TDOP^2}$$

Por tanto:

$$PDOP = \sqrt{1.9^2 + 1.2^2} \approx 2.25$$

$$GDOP = \sqrt{2.25^2 + 1.1^2} \approx 2.50$$

```python
import math

hdop = 1.9
vdop = 1.2
tdop = 1.1

pdop = math.sqrt(hdop**2 + vdop**2)
gdop = math.sqrt(pdop**2 + tdop**2)

print(f"PDOP: {pdop:.2f}")
print(f"GDOP: {gdop:.2f}")
```

Figura 59. Código 5.10: Cálculo de PDOP y GDOP.

5.7.7 Problema 7. Desplazamiento horizontal relativo

Una estación móvil presenta un desplazamiento de 1.35 m hacia el este y 0.92 m hacia el norte respecto de una posición de referencia. Determine la magnitud del desplazamiento horizontal.

Solución:

La magnitud del desplazamiento horizontal se obtiene mediante:

$$d_h = \sqrt{(\Delta E)^2 + (\Delta N)^2}$$

$$d_h = \sqrt{1.35^2 + 0.92^2} \approx 1.634\,\text{m}$$

```python
import math

dE = 1.35
dN = 0.92

desplazamiento = math.sqrt(dE**2 + dN**2)

print(f"Desplazamiento horizontal: {desplazamiento:.3f} m")
```

Figura 60. Código 5.11: Desplazamiento horizontal relativo.

5.7.8 Problema 8. Aplicación de una corrección local de coordenadas

Un punto tiene coordenadas E = 600000.00 m y N = 9600000.00 m. Si se aplica una corrección local de −5.25 m en Este y +3.10 m en Norte, determine las coordenadas corregidas.

Solución:

En este ejercicio se considera una traslación simple de coordenadas con fines didácticos:

$$E\text{corr} = E + \Delta E$$

$$N\text{corr} = N + \Delta N$$

Por tanto:

$$E\text{corr} = 600000.00 - 5.25 = 599994.75 \text{ m}$$

$$N\text{corr} = 9600000.00 + 3.10 = 9600003.10 \text{ m}$$

```
este = 600000.00
norte = 9600000.00

correccion_este = -5.25
correccion_norte = 3.10

este_corregido = este + correccion_este
norte_corregido = norte + correccion_norte

print(f"Este corregido: {este_corregido:.2f} m")
print(f"Norte corregido: {norte_corregido:.2f} m")
```

Figura 61. Código 5.12: Corrección local de coordenadas.

5.7.9 Problema 9. Diferencia de altura entre dos puntos GNSS

Una estación móvil registra una altura de 215.40 m y una estación base una altura de 210.20 m. Suponiendo que ambas alturas están referidas al mismo sistema y a la misma superficie de referencia, determine la diferencia de altura desde la base hacia la estación móvil.

Solución:

Se adopta la convención:

$$\Delta h = h\text{móvil} - h\text{base}$$

Por tanto:

$$\Delta h = 215.40 - 210.20 = 5.20 \text{ m}$$

El signo positivo indica que la estación móvil se encuentra 5.20 m por encima de la estación base.

```
        altura_movil = 215.40
        altura_base = 210.20

        delta_h = altura_movil - altura_base

        print(f"Diferencia de altura: {delta_h:.2f} m")
```

Figura 62. Código 5.13: Diferencia de altura GNSS.

5.7.10 Problema 10. Velocidad media de un recorrido

Un dron equipado con receptor GNSS recorre 752.80 m en 40.50 s. Determine la velocidad media en m/s.

Solución:

La velocidad media se calcula mediante:

$$v = \frac{d}{t}$$

$$v = \frac{752.80}{40.50} \approx 18.59\,\text{m/s}$$

```
distancia = 752.80
tiempo = 40.50

velocidad = distancia / tiempo

print(f"Velocidad media: {velocidad:.2f} m/s")
```

Figura 63. Código 5.14: Velocidad media de un recorrido.

5.8 Resumen

En este capítulo se estudiaron los fundamentos de los sistemas GNSS y su aplicación en la georreferenciación topográfica. Se revisaron los principales sistemas globales de navegación por satélite, los conceptos de coordenadas geográficas y proyectadas, el sistema UTM y la transformación entre sistemas de referencia.

También se presentaron procedimientos para organizar y transformar información espacial mediante herramientas de Python como pyproj y GeoPandas, así como ejemplos de visualización de trayectorias GNSS.

La precisión del posicionamiento GNSS depende de múltiples factores, entre ellos la geometría de los satélites, las condiciones atmosféricas, los errores orbitales y de reloj, la multitrayectoria, la calidad de las

observaciones y el método de posicionamiento utilizado. Los parámetros DOP permiten evaluar específicamente la influencia de la geometría satelital sobre la solución de posicionamiento.

El procesamiento mediante Python facilita la transformación, análisis y representación de coordenadas GNSS de forma reproducible. Sin embargo, una correcta aplicación topográfica exige identificar adecuadamente el sistema de referencia, la proyección cartográfica, la zona y el hemisferio, así como distinguir el tipo de altura utilizado en cada proyecto.

5.9 Problemas propuestos

1. Transforme las coordenadas geográficas (latitud, longitud) = $(-3.98^{\circ}, -79.10^{\circ})$ a coordenadas UTM mediante Python. Determine previamente la zona y el hemisferio correspondientes.

2. Simule una trayectoria GNSS de 10 puntos expresados como pares de latitud y longitud, y represéntela gráficamente mediante matplotlib.

3. Genere un GeoDataFrame con un conjunto de puntos en WGS 84 y transfórmelo a un sistema UTM adecuado para su ubicación.

4. Calcule la distancia total recorrida por una trayectoria GNSS después de transformar sus coordenadas geográficas a un sistema proyectado expresado en metros.

5. Simule dos trayectorias correspondientes a observaciones con dife-rente nivel de error posicional y compare gráficamente sus diferencias respecto de una trayectoria de referencia.

6. Escriba una función que determine automáticamente la zona UTM correspondiente a una longitud comprendida entre -180° y 180°.

7. Calcule la magnitud del error horizontal entre una posición observada y una posición de control conocida a partir de sus diferencias en Este y Norte.

8. Implemente una función para convertir una serie de coordenadas expresadas en grados decimales a grados, minutos y segundos, considerando correctamente valores positivos y negativos.

9. Aplique una traslación local conocida a un conjunto de coordenadas UTM y compare las coordenadas originales con las corregidas.

10. Genere un informe mediante Python que muestre las coordenadas geográficas originales, las coordenadas UTM transformadas y las diferencias planas entre puntos consecutivos.

CAPÍTULO VI

6 AJUSTE DE OBSERVACIONES TOPOGRÁFICAS POR MÍNIMOS CUADRADOS CON PYTHON

En los levantamientos topográficos y geodésicos es inevitable que las observaciones estén afectadas por errores e incertidumbres. Incluso cuando se emplean instrumentos precisos y procedimientos rigurosos, los ángulos, distancias, desniveles y coordenadas observadas pueden presentar discrepancias debidas a factores instrumentales, ambientales, metodológicos y aleatorios. Por ello, cuando existen observaciones redundantes, resulta necesario aplicar procedimientos matemáticos que permitan estimar las incógnitas y evaluar la consistencia de las mediciones.

El método de los mínimos cuadrados constituye una herramienta fundamental para el ajuste de observaciones en topografía y geodesia. Su principio consiste en determinar una solución que minimice la suma ponderada de los cuadrados de los residuos, de manera que las observaciones ajustadas sean compatibles con el modelo matemático adoptado y con la precisión relativa asignada a los datos (Mikhail & Gracie, 1981; Wolf & Ghilani, 2012).

Este método tiene aplicaciones directas en el ajuste de poligonales, redes planimétricas y altimétricas, observaciones GNSS, mediciones redundan-tes de distancias y ángulos, y en general en problemas donde sea necesario estimar parámetros a partir de observaciones con incertidumbre. Python per-mite implementar estos procedimientos de forma transparente, reproducible y eficiente.

6.1 Fundamento matemático del método

En un modelo lineal de observaciones, el ajuste por mínimos cuadrados puede expresarse mediante:

$$l + v = Ax\hat{}$$

donde:

l es el vector de observaciones;

v es el vector de residuos;

A es la matriz de diseño o matriz de coeficientes;

$x\hat{}$ es el vector de incógnitas ajustadas.

El criterio de mínimos cuadrados ponderados consiste en minimizar:

$$\Phi = vT\ Pv$$

donde P es la matriz de pesos de las observaciones. Cuando las observaciones son independientes y poseen la misma precisión relativa, puede utilizarse $P = I$, donde I es la matriz identidad. Cuando las observaciones presentan precisiones diferentes, los pesos deben definirse de acuerdo con su modelo estocástico.

A partir de este criterio se obtienen las ecuaciones normales:

$$Nx\hat{} = n$$

donde:

$$N = AT\ PA$$

y:

$$n = AT\ Pl$$

Por tanto, si la matriz normal N es no singular, la solución puede expresarse como:

$$x\hat{} = (\ AT\ PA)-1\ AT\ Pl$$

En la implementación computacional resulta preferible resolver directamente el sistema de ecuaciones normales en lugar de calcular explícitamente la matriz inversa.

Una vez obtenidas las incógnitas ajustadas, los residuos se calculan median-te:

$$v = Ax\hat{} - l$$

y las observaciones ajustadas pueden expresarse como:

$$l\hat{} = l + v = Ax\hat{}$$

Esta formulación corresponde al modelo lineal paramétrico y constituye una base para comprender ajustes topográficos de mayor complejidad.

6.2 Modelo funcional y modelo estocástico

Todo ajuste por mínimos cuadrados requiere definir adecuadamente dos componentes fundamentales:

- **Modelo funcional:** establece las relaciones matemáticas entre las observaciones y las incógnitas;

- **Modelo estocástico:** describe la incertidumbre y las relaciones estadísticas de las observaciones mediante varianzas, covarianzas y pesos.

En topografía, el modelo funcional puede representar, por ejemplo, relaciones entre coordenadas y distancias, desniveles y cotas, o ángulos y orientaciones. El modelo estocástico permite incorporar la calidad relativa de las observaciones y asignar una influencia adecuada a cada una dentro del ajuste.

Cuando se conoce la matriz de covarianzas de las observaciones, Cl , la matriz de pesos puede definirse, salvo un factor común de escala, mediante:

$$P = C_l^{-1}$$

Una definición adecuada del modelo estocástico es importante porque los resultados del ajuste y las estimaciones de precisión dependen de los pesos y correlaciones adoptados.

6.3 Ajuste por mínimos cuadrados con Python

La implementación computacional del ajuste resulta directa cuando el modelo es lineal. En Python, la biblioteca numpy permite realizar de manera eficiente las operaciones con vectores y matrices.

En el siguiente ejemplo se ajusta una misma magnitud observada tres veces con igual peso.

```python
import numpy as np

def ajuste_minimos_cuadrados(A, l, P):
N = A.T @ P @ A
n = A.T @ P @ l
x_hat = np.linalg.solve(N, n)
v = A @ x_hat - l
return x_hat, v

A = np.ones((3, 1))
l = np.array([[2.015], [2.004], [1.998]])
P = np.eye(3)

x_hat, v = ajuste_minimos_cuadrados(A, l, P)

print(f"Valor ajustado: {x_hat[0, 0]:.6f}")
print("Residuos:", v.ravel())
```

Figura 64. Código 6.1: Ajuste clásico por mínimos cuadrados.

En este caso, la incógnita corresponde a una única magnitud observada tres veces. Debido a que todas las observaciones poseen el mismo peso, el valor ajustado coincide con la media aritmética:

$$\hat{x} = 2.005667$$

Los residuos representan las diferencias que deben aplicarse a las observaciones para hacerlas compatibles con el valor ajustado.

6.4 Interpretación de resultados

El ajuste por mínimos cuadrados proporciona tanto estimaciones de las incógnitas como información para evaluar la consistencia interna del conjunto de observaciones. Entre los principales resultados se encuentran:

- **Valores ajustados:** corresponden a las estimaciones de las incógnitas obtenidas mediante el modelo y los pesos adoptados;

- **Residuos:** representan las correcciones que permiten compatibilizar las observaciones con el modelo ajustado;

- **Distribución de los residuos:** permite examinar la consistencia de las observaciones e identificar valores que requieren una revisión adicional;

- **Estimaciones de precisión:** se obtienen mediante la varianza de referencia a posteriori y la matriz de covarianzas de las incógnitas ajustadas.

Los residuos deben interpretarse conjuntamente con la precisión esperada de las observaciones, la redundancia del sistema y el modelo estocástico adoptado. Un residuo grande no constituye, por sí solo, evidencia suficiente de un error sistemático o de una equivocación en la medición.

6.5 Matriz de covarianzas y precisión del ajuste

Una vez calculados los residuos, y siempre que existan grados de libertad positivos, puede estimarse la varianza de referencia a posteriori mediante:

$$\hat{\sigma}_0^2 = \frac{v^T P v}{n - u}$$

Donde:

n es el número de observaciones;

u es el número de incógnitas;

$n - u$ corresponde al número de grados de libertad.

Para un modelo de rango completo, la matriz de covarianzas de las incógnitas ajustadas puede estimarse mediante:

$$\Sigma_{\hat{x}} = \hat{\sigma}_0^2 (A^T P A)^{-1}$$

Los elementos de la diagonal principal representan las varianzas estimadas de las incógnitas ajustadas. A partir de ellas pueden calcularse sus desviaciones estándar:

$$\sigma_{\hat{x}_i} = \sqrt{(\Sigma_{\hat{x}})_{ii}}$$

Estas cantidades permiten evaluar la precisión interna de los parámetros estimados de acuerdo con el modelo funcional y estocástico adoptado.

6.6 Aplicación a una red topográfica 2D simplificada

Considérese un caso didáctico en el que las coordenadas X y Y de un punto han sido observadas dos veces de manera independiente. El problema puede formularse como un ajuste con dos incógnitas y cuatro observaciones.

Como todas las observaciones tienen el mismo peso, los valores ajustados corresponden a las medias de cada grupo:

$$\hat{X} = \frac{100.15 + 99.95}{2} = 100.05$$

$$\hat{Y} = \frac{50.05 + 50.25}{2} = 50.15$$

Los residuos resultantes permiten comprobar cómo cada observación se relaciona con las coordenadas ajustadas. Aunque este ejemplo es simplificado, ilustra la estructura matricial empleada en problemas de ajuste con varias incógnitas y observaciones redundantes.

```
import numpy as np

A = np.array([
[1.0, 0.0],
[1.0, 0.0],
[0.0, 1.0],
[0.0, 1.0]
])

l = np.array([
[100.15],
[ 99.95],
[ 50.05],
[ 50.25]
])

P = np.eye(4)

x_hat, v = ajuste_minimos_cuadrados(A, l, P)

print(f"X ajustada: {x_hat[0, 0]:.3f}")
print(f"Y ajustada: {x_hat[1, 0]:.3f}")
print("Residuos:")
print(v)
```

Figura 65. Código 6.2: Ajuste simplificado de dos coordenadas.

6.7 Problemas resueltos

6.7.1 Problema 1. Ajuste de una distancia observada varias veces

Una distancia fue observada tres veces con los siguientes resultados: 50.124 m, 50.118 m y 50.131 m. Ajuste la distancia suponiendo igual peso en todas las observaciones.

Solución:

Se trata de un ajuste de una sola incógnita observada repetidamente. Al asignar el mismo peso a las tres observaciones, el valor ajustado corresponde a su media ponderada.

```
import numpy as np

A = np.ones((3, 1))
l = np.array([[50.124], [50.118], [50.131]])
P = np.eye(3)

x_hat, v = ajuste_minimos_cuadrados(A, l, P)

print(f"Distancia ajustada: {x_hat[0, 0]:.4f} m")
print("Residuos:", v.ravel())
```

Figura 66. Código 6.3: Ajuste de una distancia repetida.

La distancia ajustada es 50.1243 m.

6.7.2 Problema 2. Ajuste ponderado de una dirección

Una dirección fue observada tres veces con valores 38.245◦, 38.252◦ y 38.240◦. La segunda observación se considera de mejor calidad y recibe peso 2, mientras que las otras reciben peso 1. Determine el valor ajustado.

Solución:

```
import numpy as np

A = np.ones((3, 1))
l = np.array([[38.245], [38.252], [38.240]])
P = np.diag([1.0, 2.0, 1.0])

x_hat, v = ajuste_minimos_cuadrados(A, l, P)

print(f"Direccion ajustada: {x_hat[0, 0]:.5f} grados")
print("Residuos:", v.ravel())
```

Figura 67. Código 6.4: Ajuste ponderado de una dirección.

El valor ajustado es 38.24725◦.

6.7.3 Problema 3. Cálculo de la varianza de referencia a posteriori

A partir de tres observaciones de una misma magnitud, determine la varianza de referencia a posteriori del ajuste.

Solución:

Se utiliza la expresión:

$$\hat{\sigma}_0^2 = \frac{v^T P v}{n - u}$$

donde n es el número de observaciones y u el número de incógnitas.

```
import numpy as np

A = np.ones((3, 1))
l = np.array([[2.015], [2.004], [1.998]])
P = np.eye(3)

x_hat, v = ajuste_minimos_cuadrados(A, l, P)
grados_libertad = l.shape[0] - A.shape[1]
sigma0_2 = float((v.T @ P @ v)[0, 0] / grados_libertad)

print(f"Varianza a posteriori: {sigma0_2:.8f}")
```

Figura 68. Código 6.5: Varianza de referencia a posteriori.

La varianza de referencia a posteriori es aproximadamente $\hat{\sigma}^2 = 0.00007433$.

6.7.4 Problema 4. Ajuste de coordenadas planas repetidas

Se observaron dos veces las coordenadas de un punto: X = 100.18 m y 99.96 m; Y = 50.04 m y 50.21 m. Ajuste ambas coordenadas suponiendo igual peso.

Solución:

```
import numpy as np

A = np.array([[1., 0.], [1., 0.],
[0., 1.], [0., 1.]])
l = np.array([[100.18], [99.96], [50.04], [50.21]])
P = np.eye(4)

x_hat, v = ajuste_minimos_cuadrados(A, l, P)
print(f"X ajustada: {x_hat[0, 0]:.3f} m")
print(f"Y ajustada: {x_hat[1, 0]:.3f} m")
print("Residuos:", v.ravel())
```

Figura 69. Código 6.6: Ajuste de coordenadas planas.

Los resultados son: $\hat{X}$ = 100.070 m y $\hat{Y}$ = 50.125 m.

6.7.5 Problema 5. Error de cierre lineal como control previo

Una poligonal presenta componentes de cierre $fx = -0.25$ m y $fy = 0.35$ m. Calcule la magnitud del error de cierre lineal.

Solución:

Este cálculo no constituye un ajuste por mínimos cuadrados, pero representa un control previo útil para evaluar el cierre de una poligonal.

$$e = \sqrt{f_x^2 + f_y^2}$$

```
import math

fx = -0.25
fy = 0.35
error = math.sqrt(fx**2 + fy**2)

print(f"Error de cierre: {error:.3f} m")
```

Figura 70. Código 6.7: Magnitud del error de cierre.

La magnitud del error de cierre es $e \approx 0.430$ m.

6.7.6 Problema 6. Compensación angular simple por reparto uniforme

El error de cierre angular de una poligonal de 6 lados es de 15″. Determine la corrección uniforme que debería aplicarse a cada ángulo, suponiendo igual peso.

Solución:

Este procedimiento corresponde a una compensación aritmética simple y se presenta con fines didácticos. No sustituye un ajuste riguroso por mínimos cuadrados cuando las observaciones poseen precisiones diferentes o existe un modelo estocástico definido.

$$c = -\frac{15''}{6} = -2.5''$$

```
cierre = 15.0
numero_angulos = 6

correccion = -cierre / numero_angulos

print(f"Correccion por angulo: {correccion:.2f} segundos")
```

Figura 71. Código 6.8: Compensación angular simple.

6.7.7 Problema 7. Cálculo inverso entre dos puntos

Dadas las coordenadas de los puntos A(100.00, 100.00) y B(130.40, 140.25), determine la distancia horizontal y el azimut de A hacia B, medido desde el norte en sentido horario.

Solución:

Se calculan primero los incrementos: $\Delta X = XB - XA$ y $\Delta Y = YB - YA$.

La distancia y el azimut se obtienen mediante $d =$ $(\Delta X)2 + (\Delta Y)2$ y

$$\alpha = \text{atan2}(\Delta X, \Delta Y).$$

```python
import math

x1, y1 = 100.00, 100.00
x2, y2 = 130.40, 140.25

dx = x2 - x1
dy = y2 - y1
distancia = math.hypot(dx, dy)
azimut = math.degrees(math.atan2(dx, dy)) % 360.0

print(f"Distancia: {distancia:.3f} m")
print(f"Azimut: {azimut:.4f} grados")
```

Figura 72. Código 6.9: Cálculo inverso entre dos puntos.

6.7.8 Problema 8. Matriz de covarianzas de las incógnitas

A partir del ajuste de tres observaciones de una distancia, calcule la matriz de covarianzas de la incógnita ajustada.

Solución:

Se utiliza:

$$\Sigma_{\hat{x}} = \hat{\sigma}_0^2 (A^T P A)^{-1}$$

```python
import numpy as np

A = np.ones((3, 1))
l = np.array([[50.124], [50.118], [50.131]])
P = np.eye(3)

x_hat, v = ajuste_minimos_cuadrados(A, l, P)
N = A.T @ P @ A

grados_libertad = l.shape[0] - A.shape[1]
sigma0_2 = float((v.T @ P @ v)[0, 0] / grados_libertad)
Sigma_x = sigma0_2 * np.linalg.inv(N)

print("Matriz de covarianzas:")
print(Sigma_x)
```

Figura 73. Código 6.10: Matriz de covarianzas de las incógnitas.

6.7.9 Problema 9. Área de un polígono por coordenadas

Calcule el área de un polígono cerrado de cuatro vértices con coordenadas (0.00, 0.00), (4.20, 0.00), (4.20, 3.10) y (0.00, 3.10). Este procedimiento puede aplicarse posteriormente sobre coordenadas ajustadas.

Solución:

Se emplea la fórmula de coordenadas o fórmula del cordón.

```
puntos = [(0., 0.), (4.2, 0.),
(4.2, 3.1), (0., 3.1)]

area_doble = 0.0
for i in range(len(puntos)):
x1, y1 = puntos[i]
x2, y2 = puntos[(i + 1) % len(puntos)]
area_doble += x1 * y2 - x2 * y1

area = abs(area_doble) / 2
print(f"Area: {area:.3f} m^2")
```

Figura 74. Código 6.11: Área de un polígono por coordenadas.

El área calculada es 13.020 m2.

6.7.10 Problema 10. Verificación de perpendicularidad entre direcciones

Verifique si dos direcciones de 45◦ y 135◦ son perpendiculares.

Solución:

Dos direcciones son perpendiculares cuando su diferencia angular mínima es de 90◦.

```
        angulo1 = 45.0
        angulo2 = 135.0

        diferencia = abs(angulo1 - angulo2) % 180.0
        tolerancia = 1e-10

        if abs(diferencia - 90.0) < tolerancia:
        print("Las direcciones son perpendiculares")
        else:
        print("Las direcciones no son perpendiculares")
```

Figura 75. Código 6.12: Verificación de perpendicularidad.

6.8 Resumen

Este capítulo presentó el método de mínimos cuadrados como una herramienta fundamental para el ajuste de observaciones topográficas redundan-tes. Se desarrolló su formulación mediante el modelo funcional, la matriz de pesos, las ecuaciones normales y el cálculo de residuos.

También se explicó que la calidad del ajuste no depende únicamente de los valores ajustados, sino de la adecuada definición del modelo funcional y del modelo estocástico. La matriz de pesos, las varianzas y las covarianzas permiten representar la calidad relativa y las posibles correlaciones entre las observaciones.

La varianza de referencia a posteriori y la matriz de covarianzas de las incógnitas permiten evaluar la precisión interna de los parámetros estimados, siempre que el modelo posea redundancia suficiente. Estas herramientas son esenciales para el análisis de redes topográficas y geodésicas, donde la correcta representación de la incertidumbre de las observaciones influye directamente en la confiabilidad de los resultados.

6.9 Problemas propuestos

1. Ajuste tres observaciones redundantes de una distancia con igual peso y determine el valor ajustado y los residuos.

2. Ajuste una misma magnitud observada cuatro veces utilizando pesos diferentes y compare el resultado con el obtenido mediante pesos iguales.

3. Modele un conjunto simple de observaciones de coordenadas (X, Y) y realice su ajuste por mínimos cuadrados.

4. Calcule la varianza de referencia a posteriori de un ajuste con observaciones redundantes.

5. Obtenga la matriz de covarianzas y las desviaciones estándar de las incógnitas ajustadas.

6. Analice los residuos de un ajuste y determine cuáles observaciones presentan las mayores discrepancias respecto del modelo.

7. Compare los resultados obtenidos al modificar los pesos asignados a una observación discordante.

8. Simule una pequeña red topográfica linealizada con más observaciones que incógnitas y resuelva el sistema mediante mínimos cuadrados.

CAPÍTULO VII

7 FOTOGRAMETRÍA, NUBES DE PUNTOS Y VISUALIZACIÓN 3D CON PYTHON

La fotogrametría es una técnica que permite obtener información métrica, geométrica y temática de objetos o superficies a partir de imágenes. En topografía e ingeniería geomática, se utiliza para generar productos como ortomosaicos, modelos digitales de superficie, modelos digitales del terreno y nubes de puntos tridimensionales a partir de imágenes aéreas o terrestres (Wolf & Dewitt, 2000; Kraus, 2007).

El desarrollo de plataformas aéreas no tripuladas, cámaras digitales de al-ta resolución y métodos de reconstrucción tridimensional ha incrementado notablemente la disponibilidad de datos espaciales densos. Estos recursos permiten representar con gran detalle el relieve, la infraestructura y la vegetación, y facilitan aplicaciones en levantamientos topográficos, control de obras, monitoreo ambiental, inventario territorial y modelado del terreno. Python proporciona herramientas para procesar, visualizar y analizar estos productos de manera reproducible y eficiente.

7.1 Principios básicos de la fotogrametría

La fotogrametría moderna se apoya en la geometría de proyección y en la reconstrucción espacial a partir de múltiples imágenes de una misma escena. Entre sus conceptos fundamentales se encuentran:

- **Restitución:** proceso mediante el cual se obtiene información métrica y tridimensional a partir de imágenes;

- **Solape longitudinal y transversal:** superposición entre imágenes consecutivas y entre franjas de vuelo, necesaria para disponer de puntos comunes que permitan relacionar geométricamente las fotografías;

- **Par estereoscópico:** conjunto de dos imágenes superpuestas de una misma zona, obtenidas desde posiciones diferentes, que permite estimar información de profundidad;

- **Orientación interior:** establece la geometría interna de la cámara mediante parámetros como la distancia focal, el punto principal y, según el modelo empleado, los parámetros de distorsión;

- **Orientación exterior:** define la posición y orientación de la cámara respecto de un sistema de referencia mediante parámetros de traslación y rotación;

- **Puntos de control y georreferenciación:** permiten vincular el modelo fotogramétrico con un sistema de referencia conocido y evaluar su correspondencia con coordenadas obtenidas en el terreno.

En los flujos de trabajo fotogramétricos digitales es habitual emplear técnicas de Structure from Motion (SfM) para estimar simultáneamente la geometría de la escena y las posiciones de las cámaras a partir de imágenes superpuestas. Posteriormente, mediante procedimientos de

reconstrucción densa, puede generarse una nube de puntos con una mayor densidad espacial.

7.2 Nubes de puntos

Una nube de puntos es un conjunto de puntos definidos mediante coordenadas tridimensionales (X, Y, Z) que representan la geometría de una superficie, objeto o entorno. Dependiendo de la técnica de adquisición y del formato utilizado, cada punto puede contener atributos adicionales como color, intensidad, clasificación o información asociada al retorno de la señal.

Las nubes de puntos pueden obtenerse mediante diferentes tecnologías, entre ellas:

fotogrametría digital;

LiDAR (Light Detection and Ranging);

escáneres láser terrestres;

sensores de profundidad;

sistemas móviles de levantamiento.

Desde el punto de vista topográfico, las nubes de puntos constituyen una fuente de información para generar superficies, perfiles, secciones, curvas de nivel, modelos digitales y análisis volumétricos. Sin embargo, antes de utilizarlas para obtener productos topográficos es necesario considerar la calidad de los datos y, según el objetivo del trabajo, aplicar

procedimientos de filtrado, clasificación, depuración, interpolación y validación.

En particular, debe distinguirse entre los puntos que representan el terreno y aquellos correspondientes a vegetación, edificaciones u otros objetos. Esta diferenciación resulta fundamental cuando el objetivo es generar un modelo digital del terreno a partir de una nube de puntos.

7.3 Simulación de una nube de puntos en Python

Para fines didácticos, una nube de puntos puede simularse a partir de una superficie matemática y luego visualizarse con gráficos tridimensionales. A diferencia de una superficie continua, una nube de puntos debe mostrarse como un conjunto discreto de coordenadas.

Este ejemplo es útil para comprender la diferencia entre una superficie analítica y una representación discreta en forma de puntos.

7.4 Lectura de nubes de puntos reales

En aplicaciones reales, las nubes de puntos suelen almacenarse en formatos como LAS, LAZ, PLY o XYZ. Python dispone de bibliotecas como laspy, open3d y pyntcloud para leer y procesar estos archivos.

Este tipo de lectura constituye el primer paso para tareas posteriores como visualización, filtrado, clasificación, generación de MDT o análisis de alturas.

```python
import numpy as np
import matplotlib.pyplot as plt

# Crear una superficie ondulada y muestrearla como puntos
x = np.linspace(-10.0, 10.0, 80)
y = np.linspace(-10.0, 10.0, 80)
xg, yg = np.meshgrid(x, y)
zg = np.sin(np.sqrt(xg**2 + yg**2))

# Convertir la malla en nube de puntos
xp = xg.flatten()
yp = yg.flatten()
zp = zg.flatten()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xp, yp, zp, c=zp, cmap='terrain', s=4)
ax.set_title("Nube de puntos simulada")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.show()

```

Figura 76. Código 7.1: Generar y visualizar una nube de puntos simulada.

```python
import laspy

archivo = laspy.read("terreno.las")

print(f"Puntos leidos: {len(archivo.points)}")
print("Primeras coordenadas X:", archivo.x[:5])
print("Primeras coordenadas Y:", archivo.y[:5])
print("Primeras coordenadas Z:", archivo.z[:5])

```

Figura 77. Código 7.2: Lectura de archivo LAS con laspy.

```python
from pyntcloud import PyntCloud

cloud = PyntCloud.from_file("terreno.las")
cloud.plot(backend="pythreejs")

```

Figura 78. Código 7.3: Visualizacion 3D de nube de puntos con PyntCloud.

7.5 Visualización avanzada de nubes de puntos

Una nube de puntos real puede contener cientos de miles o millones de observaciones. Por ello, la visualización eficiente es una parte

fundamental del flujo de trabajo. Dependiendo del tipo de análisis, puede utilizarse matplotlib para exploración básica o herramientas más especializadas como open3d o pyntcloud.

En contextos académicos y profesionales, esta visualización resulta útil para revisar densidad de puntos, distribución espacial, anomalías y comportamiento general del relieve.

7.6 Ortomosaicos y modelos digitales

A partir de vuelos fotogramétricos y pares estereoscópicos, es posible ob-tener productos como:

- **Ortofotografía u ortomosaico:** imagen corregida geométricamente y georreferenciada.
- **Modelo Digital de Superficie (MDS o DSM):** representa elevaciones incluyendo objetos como árboles y edificaciones.
- **Modelo Digital del Terreno (MDT o DTM):** intenta representar el terreno desnudo, filtrando elementos superficiales.

Estos productos son esenciales en topografía, cartografía, análisis de pendientes, volumen de movimiento de tierras, drenaje y planificación territorial. Python puede intervenir especialmente en el postproceso: recorte de datos, clasificación, análisis estadístico, generación de curvas de nivel y visualización.

```python
import numpy as np

x = np.linspace(-10.0, 10.0, 80)
y = np.linspace(-10.0, 10.0, 80)
xg, yg = np.meshgrid(x, y)

total_puntos = xg.size
print(f"Numero total de puntos: {total_puntos}")

```

Figura 79. Código7.4: Número de puntos en una nube simulada.

```python
import numpy as np

x = np.linspace(-8.0, 8.0, 60)
y = np.linspace(-8.0, 8.0, 60)
xg, yg = np.meshgrid(x, y)
zg = np.sin(np.sqrt(xg**2 + yg**2))

elevacion_media = np.mean(zg)
print(f"Elevacion media: {elevacion_media:.4f}")

```

Figura 80. Código7.5:Elevacion media de una nube simulada.

7.7 Problemas resueltos

7.7.1 Problema 1. Número total de puntos en una nube simulada

Se genera una nube de puntos a partir de una malla regular de 80 × 80 nodos. Determine cuántos puntos tendrá la nube resultante.

Solución:

7.7.2 Problema 2. Cálculo de elevación media de una nube simulada

Calcule la elevación media de una nube de puntos generada a partir de una superficie ondulada.

Solución:

```
import numpy as np

x = np.linspace(-6.0, 6.0, 50)
y = np.linspace(-6.0, 6.0, 50)
xg, yg = np.meshgrid(x, y)
zg = np.cos(xg / 2.0) + np.sin(yg / 3.0)

z_max = np.max(zg)
z_min = np.min(zg)

print(f"Cota maxima: {z_max:.4f}")
print(f"Cota minima: {z_min:.4f}")

```

Figura 81. Código 7.6: Altura máxima y mínima de una nube.

```
import numpy as np

puntos = np.array([
[0.0, 0.0, 100.20],
[5.0, 2.0, 101.10],
[4.0, 6.0, 102.35],
[1.5, 4.5, 100.95]
])

centroide = np.mean(puntos, axis=0)
print(f"Centroide: X={centroide[0]:.3f}, Y={centroide[1]:.3f}, Z={
centroide[2]:.3f}")

```

Figura 82. Código 7.7: Centroide de una nube de puntos.

7.7.3 Problema 3. Altura máxima y mínima de una nube

Determine la cota máxima y la cota mínima en una nube de puntos simulada.

Solución:

7.7.4 Problema 4. Cálculo del centroide de una nube de puntos

Obtenga el centroide tridimensional de una nube de puntos.

Solución:

```python
import numpy as np

puntos = np.array([
[0.0, 0.0, 100.20],
[5.0, 2.0, 101.10],
[4.0, 6.0, 102.35],
[1.5, 4.5, 100.95]
])

filtrados = puntos[puntos[:, 2] >= 101.00]
print("Puntos filtrados:")
print(filtrados)

```

Figura 83. Código 7.8: Filtrado por elevación.

7.7.5 Problema 5. Filtrado por elevación

A partir de una nube de puntos, seleccione únicamente los puntos cuya elevación sea mayor o igual a 101.00 m.

Solución:

7.7.6 Problema 6. Visualización 3D de una nube de puntos

Represente una nube de puntos simulada con color asociado a la elevación.

Solución:

7.7.7 Problema 7. Generación de un perfil a partir de puntos

Considere una serie de puntos alineados y grafique su perfil de elevaciones.

Solución:

7.7.8 Problema 8. Estimación simple de densidad de puntos

En un área de 50 × 40 m se levantaron 4200 puntos. Calcule la densidad media de puntos por metro cuadrado.

Solución:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-5.0, 5.0, 40)
y = np.linspace(-5.0, 5.0, 40)
xg, yg = np.meshgrid(x, y)
zg = np.sin(xg) * np.cos(yg)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xg.flatten(), yg.flatten(), zg.flatten(), c=zg.flatten(),
cmap='viridis', s=6)
ax.set_title("Nube de puntos 3D")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.show()

```

Figura 84. Código 7.9: Visualización 3D de una nube simulada.

```
import matplotlib.pyplot as plt

distancias = [0.0, 10.0, 20.0, 30.0, 40.0]
cotas = [100.20, 101.15, 101.80, 101.10, 100.75]

plt.plot(distancias, cotas, marker='o')
plt.title("Perfil de elevaciones")
plt.xlabel("Distancia [m]")
plt.ylabel("Cota [m]")
plt.grid(True)
plt.show()

```

Figura 85. Código 7.10: Perfil de elevaciones desde puntos.

```
ancho = 50.0
largo = 40.0
n_puntos = 4200

area = ancho * largo
densidad = n_puntos / area

print(f"Densidad media: {densidad:.3f} puntos/m2")

```

Figura 86. Código 7.11: Densidad de puntos.

```
import numpy as np
import pandas as pd

puntos = np.array([
[0.0, 0.0, 100.20],
[1.0, 0.5, 100.45],
[2.0, 1.0, 100.70],
[3.0, 1.5, 100.95]
])

df = pd.DataFrame(puntos, columns=["X", "Y", "Z"])
df.to_csv("nube_puntos.csv", index=False)

print("Archivo exportado: nube_puntos.csv")

```

Figura 87. Código 7.12: Exportación de nube a CSV.

```
import numpy as np

z = np.array([100.20, 100.45, 100.70, 100.95, 100.55, 100.80])
desviacion = np.std(z, ddof=1)

print(f"Desviacion estandar de Z: {desviacion:.4f} m")

```

Figura 88. Código 7.13: Dispersión vertical.

7.7.9 Problema 9. Exportación de una nube simulada a CSV

Genere una nube de puntos simple y expórtela a un archivo CSV.

Solución:

7.7.10 Problema 10. Evaluación de dispersión vertical

Calcule la desviación estándar de las elevaciones de una nube pequeña de puntos.

Solución:

7.8 Resumen

Este capítulo abordó los fundamentos de la fotogrametría digital, la generación de nubes de puntos y su visualización tridimensional con Python. Se revisaron conceptos esenciales como restitución, solape, orientación y georreferenciación, así como el papel de las nubes de puntos en la construcción de modelos digitales y ortomosaicos. También se mostraron ejemplos de simulación, lectura, filtrado, visualización y análisis básico de datos tridimensionales, destacando el valor de Python como herramienta de apoyo en flujos de trabajo topográficos modernos.

7.9 Problemas propuestos

1. Genere una nube de puntos simulada sobre una función tridimensional distinta y visualícela.

2. Lea un archivo .las real con laspy y represente una muestra de los primeros puntos.

3. Calcule la elevación media y la desviación estándar vertical de una nube de puntos simulada.

4. Visualice un modelo digital del terreno generado a partir de una rejilla regular y asócielo a una escala de color.

5. Investigue cómo convertir una nube de puntos a un archivo .ply y visualícelo con Open3D.

6. Filtre una nube de puntos según un umbral de elevación y compare la nube original con la filtrada.

7. Obtenga el centroide de una nube tridimensional y discuta su signifi-cado geométrico.

8. Calcule la densidad de puntos de una nube en función del área cubierta.

9. Exporte una nube de puntos generada en Python a un archivo CSV o TXT estructurado.

10. Construya un perfil longitudinal a partir de una secuencia de puntos con coordenada y elevación.

CAPÍTULO VIII

8 SIG Y ANÁLISIS ESPACIAL CON PYTHON

Los Sistemas de Información Geográfica (SIG) permiten capturar, almacenar, gestionar, analizar y visualizar datos georreferenciados. En topografía, su aplicación resulta especialmente útil para integrar levantamientos de campo, cartografía, modelos digitales del terreno, imágenes georreferenciadas y capas temáticas asociadas al territorio. Mediante los SIG es posible realizar operaciones de análisis espacial como delimitación de zonas, cálculo de pendientes, generación de áreas de influencia, intersecciones, clasificación del relieve y evaluación de relaciones espaciales entre objetos geográficos (Burrough & McDonnell, 1998; Longley et al., 2015).

Python constituye una herramienta de gran utilidad en el ámbito de los SIG gracias a bibliotecas como geopandas, shapely, rasterio, numpy y folium. Estas permiten automatizar procesos repetitivos, procesar conjuntos de datos espaciales, integrar información vectorial y ráster, y generar productos cartográficos y resultados analíticos de manera reproducible.

8.1 Conceptos fundamentales

En SIG, la representación de la información espacial se basa principalmente en dos grandes modelos de datos:

- **Datos vectoriales:** representan entidades discretas mediante puntos, líneas y polígonos. Se utilizan, por ejemplo, para

estaciones topográficas, caminos, ríos, parcelas, límites administrativos o zonas de protección.

- **Datos ráster:** representan variables continuas a partir de una malla de celdas o píxeles. Son apropiados para elevación, temperatura, humedad, pendiente, cobertura del suelo e imágenes satelitales.

```
import geopandas as gpd

gdf = gpd.read_file("zonas_protegidas.shp")

print(gdf.head())
print(gdf.crs)

gdf.plot(column="categoria", legend=True)
```

Figura 89. Código 8.1: Cargar shapefile y mostrar información.

Otro concepto fundamental es el CRS (CoordinateReferenceSystemo

Sistema de referencia de coordenadas), que define cómo se expresan las

Coordenadas y cómo se relacionan con la superficie terrestre. Un CRS incluye datum, elipsoide y, en su caso ,proyección cartográfica. Entopografía aplicada, trabajar con el CRS correcto es indispensable para evitar errores en distancias, áreas, superposiciones y análisis espaciales.

8.2 Lectura de datos vectoriales con GeoPandas

La biblioteca GeoPandas permite leer y manipular capas vectoriales en formatos como shapefile, GeoJSON y geopackage. Es una herramienta

muy útil para el análisis espacial en topografía, ya que integra geometría y atributos en una estructura tabular de fácil manejo.

Este procedimiento permite revisar rápidamente el contenido de una capa, su sistema de referencia y la distribución temática de sus atributos.

8.3 Análisis espacial: buffers e intersecciones

Una de las operaciones más comunes en SIG es la generación de buffers, es decir, zonas de influencia alrededor de geometrías. En topografía aplicada, estas zonas se utilizan para evaluar proximidad a caminos, ríos, líneas eléctricas, áreas protegidas o zonas de riesgo.

Es importante recordar que los buffers deben calcularse en un sistema proyectado en metros y no en coordenadas geográficas, para que la distancia tenga sentido físico.

```
import geopandas as gpd

caminos = gpd.read_file("vias.shp")
caminos = caminos.to_crs(epsg=32717)

buffer = caminos.buffer(500.0)

gdf_buffer = gpd.GeoDataFrame(geometry=buffer, crs=caminos.crs)
gdf_buffer.plot()
```

Figura 90. Código 8.2: Crear un buffer de 500 m alrededor de caminos.

```
import geopandas as gpd

zonas = gpd.read_file("zonas_protegidas.shp").to_crs(gdf_buffer.crs)

intersec = gpd.overlay(gdf_buffer, zonas, how="intersection")
intersec.plot()
```

Figura 91. Código 8.3: Intersección con zonas protegidas.

Una vez generado el buffer, puede analizarse su relación con otras capas mediante operaciones de intersección, diferencia o unión espacial.

La intersección permite determinar qué parte de las áreas protegidas se encuentra dentro de la zona de influencia de las vías analizadas.

8.4 Análisis ráster con Rasterio

Los datos ráster son fundamentales en topografía, especialmente cuando se trabaja con modelos digitales del terreno, ortomosaicos, mapas de pendientes o coberturas temáticas. La biblioteca rasterio permite leer estos archivos y operar sobre ellos de manera eficiente.

```
import rasterio
from rasterio.plot import show

with rasterio.open("mdt.tif") as src:
show(src, cmap="terrain")
```

Figura 92.Código 8.4: Leer raster y graficar mapa de elevación.

```
import numpy as np
from matplotlib import pyplot as plt
import rasterio

with rasterio.open("mdt.tif") as src:
elev = src.read(1).astype("float64")
transform = src.transform

dx = transform.a
dy = abs(transform.e)

grad_y, grad_x = np.gradient(elev, dy, dx)
slope = np.sqrt(grad_x**2 + grad_y**2)

plt.imshow(slope, cmap="viridis")
plt.title("Mapa de Pendientes")
plt.colorbar(label="Pendiente [m/m]")
plt.show()
```

Figura 93. Código 8.5: Calculo de pendiente con NumPy.

El trabajo con ráster exige especial cuidado con la resolución espacial, el sistema de referencia, los valores nulos y el significado físico de cada celda.

8.5 Cálculo de pendientes y orientación

A partir de un MDT ráster es posible estimar la pendiente del terreno calculando el gradiente espacial de la elevación. Este análisis es muy importante en obras viales, drenaje, estabilidad de taludes, escorrentía superficial y planificación territorial.

Para que el cálculo sea correcto, es necesario considerar el tamaño del píxel en las direcciones x e y, obtenido a partir de la transformación espacial del ráster.

Este resultado puede transformarse a porcentaje o grados según el objetivo del análisis.

8.6 Mapas interactivos con Folium

Cuando se desea comunicar información espacial en entornos web o gene-rar visualizaciones interactivas de apoyo, la biblioteca folium ofrece una alternativa sencilla y eficaz. Esto resulta útil para mostrar puntos GNSS, rutas, zonas de influencia o capas temáticas sobre mapas base.

Este tipo de visualización no reemplaza al análisis topográfico riguroso, pero constituye una herramienta muy útil para exploración preliminar, difusión de resultados y apoyo a la toma de decisiones.

```python
import folium

mapa = folium.Map(location=[-3.99, -79.12], zoom_start=14)
folium.Marker([-3.99, -79.12], popup="Punto 1").add_to(mapa)

mapa.save("mapa_interactivo.html")
```

Figura 94. Código 8.6: Mapa web con puntos GNSS.

```python
import geopandas as gpd

cuenca = gpd.read_file("cuenca.shp")
cuenca = cuenca.to_crs(epsg=32718)

area_ha = cuenca.geometry.area.sum() / 10000.0
print(f"Area total: {area_ha:.2f} ha")
```

Figura 95. Código 8.7: Area de una cuenca.

8.7 Problemas resueltos

8.7.1 Problema 1. Cálculo del área de una cuenca

Calcule el área de una cuenca a partir de un shapefile utilizando geopandas.

Solución:

Para calcular áreas correctamente, la capa debe estar en un sistema proyec-tado en metros.

8.7.2 Problema 2. Buffer alrededor de un río

Genere un buffer de 300 m alrededor de un río.

Solución:

```
import geopandas as gpd

rio = gpd.read_file("rio.shp")
rio = rio.to_crs(epsg=32718)

buffer = rio.buffer(300.0)
buffer_gdf = gpd.GeoDataFrame(geometry=buffer, crs=rio.crs)

buffer_gdf.to_file("buffer_rio.shp")
```

Figura 96. Código 8.8: Buffer de rio.

```
import geopandas as gpd

cultivos = gpd.read_file("cultivos.shp")
cultivos = cultivos.to_crs(buffer_gdf.crs)

intersectan = gpd.overlay(cultivos, buffer_gdf, how="intersection")
print(f"Cultivos afectados: {len(intersectan)}")
```

Figura 97. Código 8.9: Intersección con cultivos.

8.7.3 Problema 3. Intersección con cultivos

Determine cuántos cultivos intersectan el buffer del río.

Solución:

8.7.4 Problema 4. Lectura de MDT y cálculo de pendiente

Lea un MDT y genere un mapa básico de pendientes.

Solución:

8.7.5 Problema 5. Clasificación del relieve

Clasifique la pendiente en tres categorías: baja ($< 5^\circ$), media (5° a 15°) y alta ($> 15^\circ$).

Solución:

Primero se convierte la pendiente a grados y luego se clasifica.

```
import numpy as np
import rasterio
from matplotlib import pyplot as plt

with rasterio.open("mdt.tif") as src:
elev = src.read(1).astype("float64")
perfil = src.profile
transform = src.transform

dx = transform.a
dy = abs(transform.e)

grad_y, grad_x = np.gradient(elev, dy, dx)
pendiente = np.sqrt(grad_x**2 + grad_y**2)

plt.imshow(pendiente, cmap="viridis")
plt.colorbar(label="Pendiente [m/m]")
plt.title("Mapa de pendientes")
plt.show()
```

Figura 98. Código 8.10: Mapa de pendientes.

```
import numpy as np
import rasterio

with rasterio.open("pendiente.tif") as src:
pendiente = src.read(1).astype("float32")
perfil = src.profile

clasificado = np.select(
[pendiente < 5.0, (pendiente >= 5.0) & (pendiente <= 15.0), pendiente >
15.0],
[1, 2, 3]
).astype("uint8")

perfil.update(dtype="uint8", count=1)

with rasterio.open("relieve.tif", "w", **perfil) as dst:
dst.write(clasificado, 1)
```

Figura 99. Código 8.11: Clasificación de relieve.

```
import geopandas as gpd

puntos = gpd.read_file("puntos_gnss.shp")
puntos.to_file("puntos_gnss.geojson", driver="GeoJSON")
```

Figura 100. Código 8.12: Exportar puntos a GeoJSON.

```
        import folium
        import geopandas as gpd

        puntos = gpd.read_file("puntos_gnss.geojson")
        m = folium.Map(location=[-1.50, -78.50], zoom_start=13)

        for _, row in puntos.iterrows():
        coords = row.geometry.coords[0]
        folium.Marker([coords[1], coords[0]], popup=row["nombre"]).add_to(m)

        m.save("mapa_gnss.html")
```

Figura 101. Código 8.13: Mapa web con folium.

8.7.6 Problema 6. Exportación a GeoJSON

Exporte un conjunto de puntos GNSS a formato GeoJSON.

Solución:

8.7.7 Problema 7. Visualización con Folium

Cree un mapa web con puntos GNSS usando folium.

Solución:

8.7.8 Problema 8. Conteo de parcelas por categoría

Calcule cuántas parcelas existen por tipo de cultivo.

Solución:

8.7.9 Problema 9. Extracción de valores de ráster

Extraiga valores de elevación del MDT para puntos GNSS.

Solución:

8.7.10 Problema 10. Unión espacial entre capas

Realice una unión espacial entre parcelas y zonas administrativas.

Solución:

```
import geopandas as gpd

cultivos = gpd.read_file("cultivos.shp")
conteo = cultivos["tipo"].value_counts()

print(conteo)
```

Figura 102. Código 8.14: Conteo por categoría.

```
from rasterstats import point_query

valores = point_query("puntos_gnss.shp", "mdt.tif")
print(valores)
```

Figura 103. Código 8.15: Extracción de elevaciones.

8.8 Resumen

Este capítulo mostró cómo Python puede utilizarse para realizar análisis espaciales avanzados sobre datos vectoriales y ráster, integrando herramientas propias de los SIG modernos al trabajo topográfico. Se trabajó con lectura de capas, cálculo de áreas, buffers, intersecciones, pendientes, clasificación del relieve, extracción de valores ráster y mapas interactivos. Todo ello evidencia que Python constituye un entorno robusto y versátil para integrar análisis espacial, cartografía digital y topografía aplicada.

```
        import geopandas as gpd

        parcelas = gpd.read_file("parcelas.shp")
        zonas = gpd.read_file("zonas.shp").to_crs(parcelas.crs)

        union = gpd.sjoin(parcelas, zonas, how="left", predicate="intersects")
        union.to_file("parcelas_zonas.shp")
```

Figura 104. Código 8.16: Union espacial.

8.9 Problemas propuestos

1. Cargue un shapefile de cuencas hidrográficas y calcule su área total en hectáreas.

2. Genere un buffer de 300 m alrededor de un río y determine cuántos cultivos lo intersectan.

3. Lea un MDT y calcule un mapa de pendientes considerando el tamaño de píxel.

4. Clasifique el relieve en tres categorías según la pendiente.

5. Genere un mapa web con marcadores para cinco puntos GNSS usando folium.

6. Realice una unión espacial entre parcelas y zonas administrativas y analice cuántas parcelas quedan en cada zona.

7. Extraiga valores de elevación de un ráster para un conjunto de puntos de control.

8. Compare el resultado de un análisis de buffer en coordenadas geográficas y en coordenadas proyectadas, e interprete la diferencia.

RESUMEN FINAL

Este libro presenta un recorrido integral por los fundamentos de la topografía moderna desde un enfoque computacional apoyado en Python. A lo largo de sus capítulos se desarrolla una secuencia progresiva que integra fundamentos teóricos, formulación técnica, automatización de cálculos y visualización de resultados.

En primer lugar, se introduce Python como herramienta de apoyo para aplicaciones topográficas, mediante el estudio de tipos de datos, operadores, estructuras de control, funciones y módulos. Posteriormente, se aborda la planimetría a partir del manejo de coordenadas planas, direcciones, distancias, poligonales y errores de cierre, mostrando procedimientos para automatizar cálculos geométricos fundamentales.

La altimetría incorpora la dimensión vertical del terreno mediante el análisis de cotas, desniveles, curvas de nivel, perfiles y modelos digitales del terreno. Sobre esta base se desarrolla el cálculo de volúmenes mediante métodos de secciones transversales, prismoidal, cuadrícula y diferencia de superficies, con aplicaciones orientadas a la estimación de cortes y rellenos. A continuación, se estudian el posicionamiento GNSS y la georreferenciación, considerando sistemas de coordenadas, transformaciones y procedimientos básicos para el tratamiento de información espacial.

El ajuste de observaciones por mínimos cuadrados introduce procedimientos para el tratamiento de mediciones redundantes, el análisis de residuos y la estimación de la precisión de los resultados

ajustados. Posteriormente, la fotogrametría y las nubes de puntos amplían el análisis hacia la reconstrucción tridimensional, la visualización 3D y el procesamiento de datos espaciales densos. Finalmente, los Sistemas de Información Geográfica in-tegran operaciones de análisis vectorial y ráster, permitiendo relacionar la información topográfica con diferentes formas de representación y análisis del territorio.

De manera transversal, Python constituye el eje computacional del libro, ya que permite automatizar procedimientos, verificar resultados, procesar información espacial y representar datos topográficos. En conjunto, los contenidos proporcionan una base teórica y práctica para estudiantes, docentes y profesionales interesados en incorporar herramientas computacionales en el aprendizaje y la aplicación de la topografía contemporánea.

CONCLUCIONES

- El desarrollo del libro permitió integrar los fundamentos de la topografía con procedimientos computacionales aplicados, mostrando que la programación puede complementar de manera efectiva los métodos tradicionales de cálculo, análisis y representación de información territorial.

- Python se utilizó como herramienta transversal para automatizar procesos en planimetría, altimetría, cálculo de volúmenes, georreferenciación, ajuste de observaciones, fotogrametría, procesamiento de nubes de puntos y análisis espacial mediante SIG.

- La incorporación de rutinas computacionales favorece la reproducibilidad de los cálculos, la verificación de resultados y el tratamiento sistemático de datos, sin sustituir el criterio técnico necesario para interpretar observaciones, seleccionar métodos y evaluar la calidad de la información obtenida.

- Los ejemplos y problemas desarrollados muestran que la combinación entre fundamentos topográficos y programación facilita la comprensión de los procedimientos, permite explorar diferentes escenarios de cálculo y fortalece la capacidad de análisis aplicada a situaciones de carácter académico y profesional.

- Los contenidos abordados proporcionan una base para continuar el estudio de áreas relacionadas con la geomática, la geodesia, la cartografía digital, la fotogrametría, los Sistemas de Información Geográfica y el análisis espacial.

- En conjunto, el libro constituye una guía de apoyo para estudiantes, docentes y profesionales interesados en incorporar herramientas computacionales al aprendizaje y a la práctica de la topografía, manteniendo como principio fundamental la correcta interpretación técnica de los datos y de los resultados obtenidos.

BIBLIOGRAFÍA

Burrough, P. A., & McDonnell, R. A. (1998). Principles of geographical information systems. Oxford University Press.

Ghilani, C. D. (2021). Elementary surveying: An introduction to geomatics

(16th ed.). Pearson.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Kavanagh, B. F., & Slattery, T. J. (2014). Surveying: Principles and appli-cations (9th ed.). Pearson.

Kraus, K. (2007). Photogrammetry: Geometry from images and laser scans.

Walter de Gruyter.

Longley, P. A., Goodchild, M. F., Maguire, D. J., & Rhind, D. W. (2015).

Geographic information systems and science (4th ed.). Wiley.

McKinney, W. (2022). Python for data analysis: Data wrangling with pan-das, NumPy, and Jupyter (3rd ed.). O'Reilly Media.

Mikhail, E. M., & Gracie, G. (1981). Analysis and adjustment of survey measurements. Van Nostrand Reinhold.

Van Rossum, G., & Drake, F. L. (2009). Python 3 reference manual. Crea-teSpace.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cour-napeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., & Nelson, A. R. J. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Wolf, P. R., & Dewitt, B. A. (2000). Elements of photogrammetry with applications in GIS (3rd ed.). McGraw-Hill.

Wolf, P. R., & Ghilani, C. D. (2012). Elementary surveying: An introduction to geomatics (13th ed.). Pearson.

Cálculo topográfico con Python guía de prácticas y ejemplos, se publicó en el mes de noviembre de 2026.

ISBN: 978-9907-802-50-4

Grupo Editorial BLR
Ecuador
Cel: +593 98 320 4362
https://grupoblr.com/
publicaciones@grupoblr.com

BIOGRAFÍA DEL AUTOR

Javier Oswaldo Yunga Márquez:

Magíster en Ingeniería en Vialidad y Transportes Ingeniero Civil con sólida trayectoria docente en la Universidad Técnica de Machala. Es Magíster en Ingeniería en Vialidad y Transportes, y cuenta con experiencia en áreas como topografía, diseño geométrico y sistemas de información geográfica, integrando teoría con práctica profesional.

Desde el año 2014, ha prestado sus servicios en diversas unidades académicas y facultades, como Ingeniería Civil y Ciencias Agropecuarias, mediante contratos ocasionales docentes. Su compromiso con la formación de ingenieros se refleja en su labor continua, abarcando asignaturas técnicas y aplicadas con un enfoque moderno.

Su enseñanza se apoya en herramientas como Python, QGIS y software especializado, fomentando el aprendizaje activo y contextualizado. Ha participado en procesos de fortalecimiento académico y desarrollo de contenidos aplicados a problemas reales del entorno local.

Con enfoque ético y compromiso institucional, promueve el pensamiento analítico y el uso de tecnologías para resolver desafíos técnicos y sociales en el ámbito de la ingeniería.

CÁLCULO TOPOGRÁFICO CON PYTHON GUÍA DE PRÁCTICAS Y EJEMPLOS

Estimado lector, Esta obra técnico-pedagógica articula los fundamentos geométricos, trigonométricos y estadísticos de la topografía moderna con las capacidades de automatización, procesamiento numérico y visualización del lenguaje de programación Python. El texto parte de la premisa de que la programación no sustituye la comprensión teórica ni el criterio técnico del ingeniero en la interpretación de observaciones de campo, sino que actúa como un entorno reproducible para optimizar operaciones repetitivas, verificar resultados y modelar datos geoespaciales complejos.

Estructurado en ocho capítulos progresivos que combinan formulación matemática, código implementado y problemas resueltos, el volumen inicia con la sintaxis esencial de Python y el ecosistema de bibliotecas científicas (numpy, pandas, matplotlib y scipy) aplicadas a operaciones geométricas elementales. Posteriormente, aborda la planimetría mediante el cálculo de incrementos de coordenadas por azimut y distancia y el control de errores de cierre en poligonales, seguido del estudio de la altimetría, el trazado de perfiles, la interpolación de curvas de nivel y la construcción tridimensional de Modelos Digitales del Terreno (MDT). Sobre esta base, desarrolla la cuantificación de movimientos de tierra (corte y relleno) mediante los métodos de secciones transversales, prismoidal, cuadrícula y diferencia de superficies. En su segunda mitad, la obra profundiza en la georreferenciación y transformación de coordenadas GNSS mediante pyproj y geopandas, el ajuste estadístico riguroso de observaciones redundantes por mínimos cuadrados ponderados, el procesamiento y visualización 3D de nubes de puntos fotogramétricas y LiDAR (laspy, pyntcloud), y el análisis espacial vectorial y ráster en Sistemas de Información Geográfica con rasterio, shapely y folium.

En su conjunto, el libro se consolida como un manual formativo y de consulta práctica indispensable para estudiantes, docentes y profesionales de ingeniería civil, geomática, topografía y ciencias de la Tierra, dotándolos de competencias computacionales sólidas para enfrentar los desafíos del modelado territorial y el control geométrico de obras contemporáneas.

GRUPO BLR

Grupo Editorial BLR
Ecuador
Cel: +593 98 320 4362
https://grupoblr.com/
publicaciones@grupoblr.com

ISBN: 978-9907-802-50-4

9 789907 802504

UNIVERSIDAD TÉCNICA DE MACHALA 1969