Proyecto final

1. Ciclo de vida del software: este sirve para guiar a los equipos de desarrollo en la planificación, creación, implementación y mantenimiento del software de manera organizada y eficiente, se identificarán etapas hasta llegar a las pruebas.

1.1 Planificación y análisis

- Descripción general sistema:

A través de la obtención de datos crudos desde una radiación en terreno es que se comienza la exportación de estos para las distintas actividades que puede realizar un sistema enfocado a la obtención de los resultados de este levantamiento a través de formatos teóricos y gráficos. Esto echo a través de procesos ordenados donde principalmente se realizan las especificaciones de requerimiento exponiendo la funcionalidad del sistema y distintos diagramas que facilitan el entendimiento de estos para poder finalmente ver si estas precondiciones se cumplen o no a través de la ejecución del sistema.

- Objetivo principal:

En este sistema se quiere lograr como objetivo principal automatizar el proceso de cálculo aplicando preprocesamientos para los datos crudos, algoritmos y técnicas de cálculo topográfico, ofreciendo al usuario herramientas de visualización y análisis que permitan obtener los resultados requeridos.

- Herramientas a utilizar:

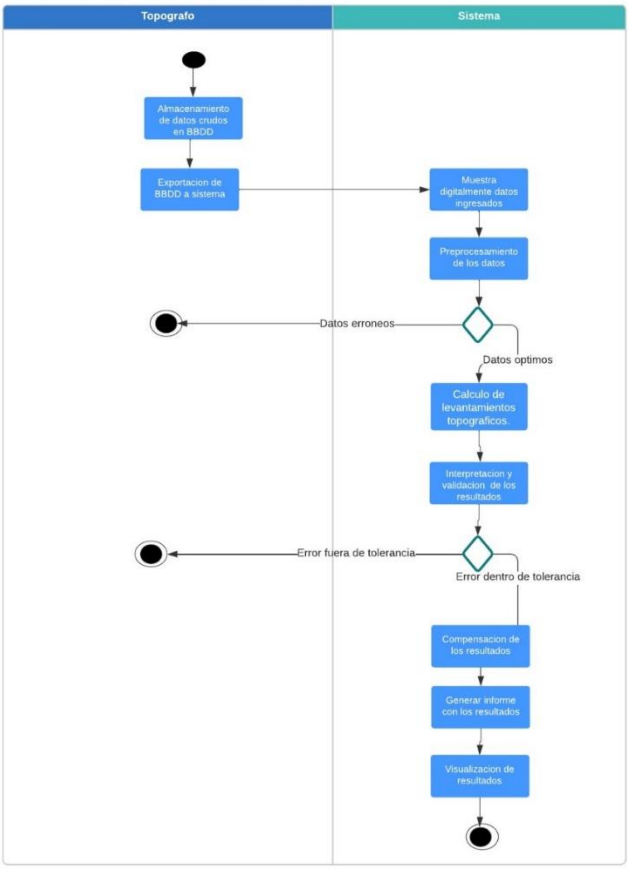
-Lucidchat: Aunque es más conocido como una herramienta de diagramación en general, también se puede considerar como una herramienta CASE ya que se puede acudir a la creación de diagramas UML proporcionando para estos ejemplos y símbolos estandarizados en este caso se utilizara para la realización de los diferentes diagramas que dan una mejor explicación del proceso que damos a entender.

-Xampp control panel: esta es una herramienta de administración que permite a los usuarios gestionar fácilmente servidores locales para el desarrollo web. En este caso se utilizará MySQL para la creación de la base de datos utilizada.

-Matlab: es una plataforma de cálculo numérico y un lenguaje de programación desarrollado por MathWorks, ampliamente utilizado en ingeniería, ciencias y economía. Ofrece un entorno interactivo para el desarrollo de algoritmos, visualización de datos, análisis de datos y computación numérica. En este caso será utilizada para la realización de pruebas de nuestro sistema y comprobación de su funcionamiento.

1.2 Diseño y arquitectura

- Diagrama de actividades:



Dentro de este se pueden identificar

-Requerimientos funcionales

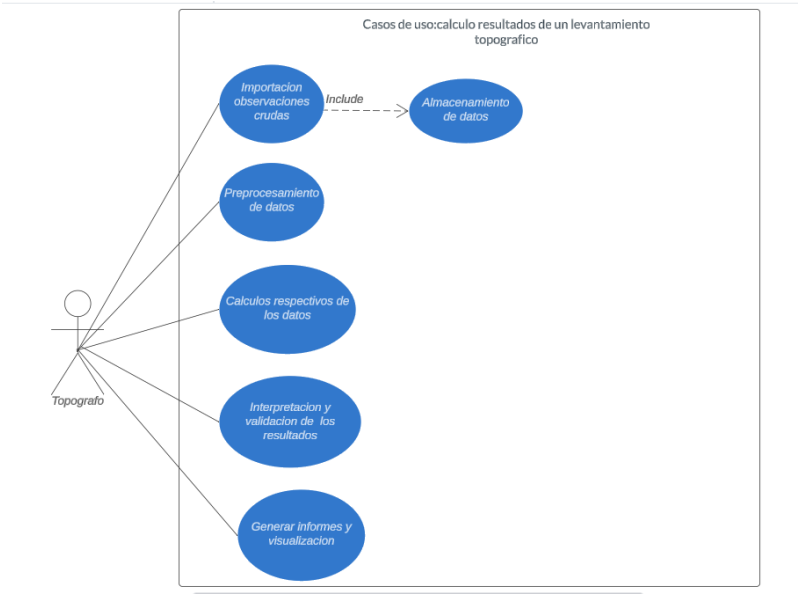
- Almacenamiento de datos crudos en BBDD
- Exportación y muestra de BBDD a sistema
- Preprocesamiento de los datos
- Cálculo de levantamientos topográficos
- Interpretación y validación de los resultados
- Compensación de los resultados
- Generar informe con los resultados y su visualización

-Requerimientos no funcionales

- Rendimiento: El sistema debe ser capaz de procesar grandes volúmenes de datos de manera eficiente, asegurando tiempos de respuesta adecuados en cada etapa del flujo de trabajo.

- Seguridad: Los datos almacenados y procesados deben estar protegidos contra accesos no autorizados. Además, las interfaces de usuario y de sistema deben contar con medidas de autenticación y autorización.
- Usabilidad: La interfaz del sistema debe ser intuitiva y fácil de usar para los topógrafos, facilitando la entrada de datos y la interpretación de los resultados.

- Diagrama casos de uso y especificaciones



Función	Importación observaciones crudas
Descripción	El usuario importa las observaciones crudas digitales en un formato *.txt* desde la estación total al sistema.
Actor	Topógrafo
Precondición	El usuario debe tener acceso a los datos de nivelación.
Flujo normal	1. El usuario inicia el proceso de importación de observaciones crudas. 2. el sistema recibe los datos en formato .txt.
Flujo Alternativo	3.1 Si los datos no son válidos, el sistema muestra un mensaje de error.
Postcondición	Los datos quedan disponibles para el preproceso.

Función	Almacenamiento de datos
Descripción	El sistema realiza el almacenamiento en una base de datos.
Actor	Sistema
Precondición	Los datos tienen que estar importados
Flujo normal	1. Los datos importados son ingresados en la base de datos. 2. Los datos se almacenan en bases de datos relacionales (SQL).
Flujo Alternativo	----------

Postcondición	Los datos importados están almacenados y listos para su análisis y procesamiento.

Función	Preprocesar datos
Descripción	El sistema realiza el preprocesamiento de los datos crudos para corregir errores y prepararlos para el cálculo.
Actor	Sistema
Precondición	El Topógrafo debe haber importado los datos crudos
Flujo normal	1. El sistema realiza la limpieza de datos, eliminando datos erróneos o inconsistentes. 2. Se aplican correcciones de instrumento y otras. 3. Los datos son convertidos a unidades estándar necesarias para el análisis.
Flujo Alternativo	----------
Postcondición	Los datos preprocesados quedan listos para realizar los respectivos cálculos.

Función	Cálculos respectivos de los datos
Descripción	Se proceden a realizar el análisis para el posterior calculo con los datos.
Actor	Sistema
Precondición	Los datos tienen que estar preprocesados.
Flujo normal	1. Se Inicializar matriz para resultados 2. Se transforma de coordenadas polares a cartesianas. 3. Se aplican la conversión de ángulos a radianes 4. Se calculan coordenadas en el plano horizontal
Flujo Alternativo	----------
Postcondición	Los cálculos están realizados listo para la validación e interpretación de estos.

Función	Interpretación y validación de los resultados.
Descripción	Se interpretan los resultados y se validan como manera de comprobación.
Actor	Sistema
Precondición	Se realizaron los cálculos
Flujo normal	1. Comprobar que las coordenadas no son NaN y están dentro de un rango esperado. 2. Calcular errores y tolerancia
Flujo Alternativo	----------
Postcondición	Los resultados validados y contextualizados están listos para ser reportados.

Función	Generación de reportes.
Descripción	Se generan reportes en el formato deseado para poder así visualizar de forma ordenada los resultados.
Actor	Sistema
Precondición	Los resultados han sido interpretados y validados.
Flujo normal	1. Se crea un informe detallado que describe los datos recopilados. 2. Se presentan los resultados con gráficos y tablas explicativas.
Flujo Alternativo	----------
Postcondición	----------

- Diagrama de flujo:

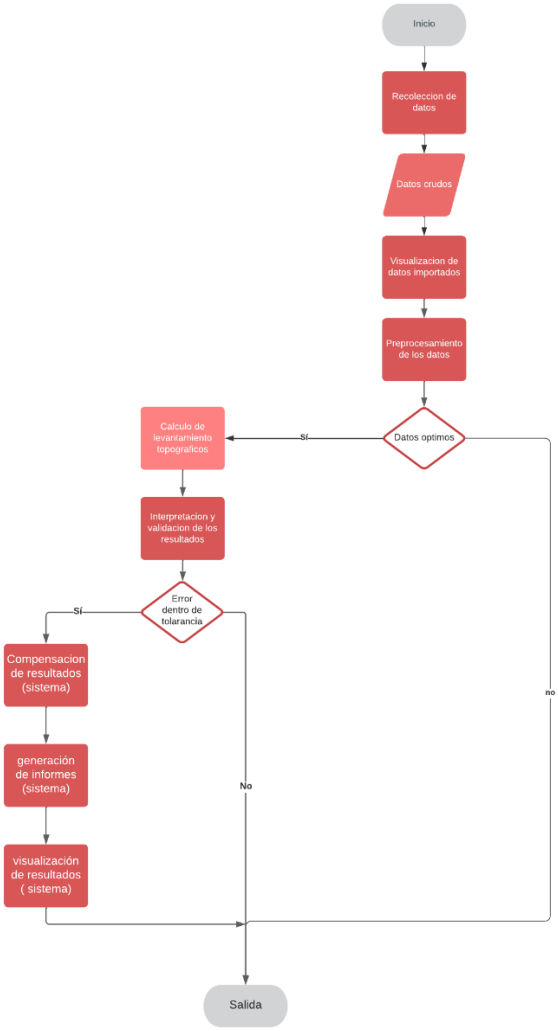
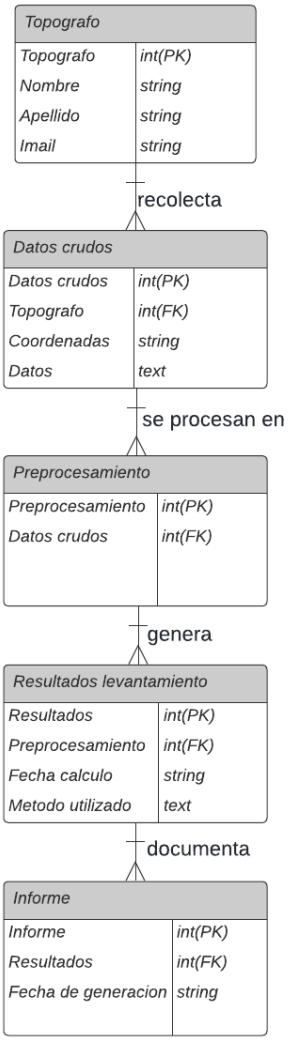


Diagrama modelo relacional:



1.3 Codificación

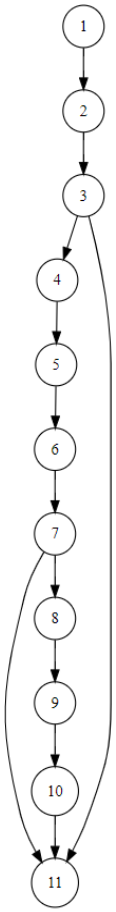
```
%% Paso 1: Lectura del archivo exportado
data = readtable('datos_crudos.csv'); % Cambia 'datos_crudos.csv' por el nombre de tu archivo

%% Paso 2: Preprocesamiento de datos crudos
% Renombrar las columnas para facilitar el acceso
data.Properties.VariableNames = {'Punto', 'Distancia', 'HD', 'VD', 'X_este', 'Y_norte', 'Z_cota', 'Cod'};

% Convertir variables necesarias a números
data.Distancia = str2double(strrep(data.Distancia, ',', '.'));
data.HD = str2double(strrep(data.HD, ',', '.'));
data.VD = str2double(strrep(data.VD, ',', '.'));
data.X_este = str2double(strrep(data.X_este, ',', '.'));
data.Y_norte = str2double(strrep(data.Y_norte, ',', '.'));
data.Z_cota = str2double(strrep(data.Z_cota, ',', '.'));

% Identificar filas inválidas y eliminarlas
filas_invalidas = any(isnan(data{:, {'Distancia', 'HD', 'VD', 'X_este', 'Y_norte', 'Z_cota'}}), 2);
data = data(~filas_invalidas, :);

%% Paso 3: Cálculos de levantamientos topográficos (Método de Radiación)
% Suponiendo que la estación está en (X0, Y0, Z0)
X0 = 0; % Coordenada X de la estación (cambia según tu punto de referencia)
Y0 = 0; % Coordenada Y de la estación (cambia según tu punto de referencia)
Z0 = 0; % Coordenada Z de la estación (cambia según tu punto de referencia)

% Inicializar matriz para resultados
resultados = zeros(height(data), 4); % [Punto, X_calc, Y_calc, Z_calc]

for i = 1:height(data)
    % Coordenadas polares a cartesianas
    Distancia = data.Distancia(i);
    HD = data.HD(i);
    VD = data.VD(i);

    % Convertir ángulos a radianes
    HD_rad = deg2rad(HD);
    VD_rad = deg2rad(VD);

    % Calcular coordenadas en el plano horizontal
    X_calc = X0 + Distancia * cos(VD_rad) * cos(HD_rad);
    Y_calc = Y0 + Distancia * cos(VD_rad) * sin(HD_rad);
    Z_calc = Z0 + Distancia * sin(VD_rad);

    % Almacenar resultados
    resultados(i, :) = [i, X_calc, Y_calc, Z_calc];
end

%% Paso 4: Interpretación y validación de resultados
% Validación simple: comprobar que las coordenadas no son NaN y están dentro de un rango esperado
valido = all(~isnan(resultados(:, 2:4)), 2) & resultados(:, 2) >= -10000 & resultados(:, 2) <= 10000 & ...
         resultados(:, 3) >= -10000 & resultados(:, 3) <= 10000 & resultados(:, 4) >= -10000 & resultados(:,
4) <= 10000;

% Filtrar resultados no válidos (opcional)
resultados_validos = resultados(valido, :);

% Calcular errores y tolerancia
errores = sqrt((resultados_validos(:, 2) - data.X_este).^2 + (resultados_validos(:, 3) - data.Y_norte).^2 +
(resultados_validos(:, 4) - data.Z_cota).^2);
tolerancia = 1000; % Define tu propia tolerancia aquí

% Mostrar errores y puntos fuera de tolerancia
disp('Errores:');
disp(errores);
puntos_fuera_tol = errores > tolerancia;
if any(puntos_fuera_tol)
    warning('Algunos puntos están fuera de tolerancia.');
    disp('Puntos fuera de tolerancia:');
    disp(array2table(resultados_validos(puntos_fuera_tol, :), 'VariableNames', {'Punto', 'X_este', 'Y_norte',
'Z_cota'}));
else
```

```
    disp('Todos los puntos están dentro de tolerancia.');
end

%% Paso 5: Generación de informe
% Guardar resultados en un archivo .csv
results_table = array2table(resultados_validos, 'VariableNames', {'Punto', 'X_este', 'Y_norte', 'Z_cota'});
writetable(results_table, 'resultados_levantamientos.csv');

% Generar informe
informe_filename = 'informe_levantamientos.txt';
fid = fopen(informe_filename, 'w');
fprintf(fid, 'Informe de Levantamientos Topográficos\n\n');

fprintf(fid, 'Errores:\n');
for i = 1:length(errores)
    fprintf(fid, 'Punto %d: %.2f\n', resultados_validos(i, 1), errores(i));
end

fprintf(fid, '\n');
if any(puntos_fuera_tol)
    fprintf(fid, 'Algunos puntos están fuera de tolerancia.\n');
    fprintf(fid, 'Puntos fuera de tolerancia:\n');
    for i = 1:size(resultados_validos(puntos_fuera_tol, :), 1)
        fprintf(fid, 'Punto %d: X=%.2f, Y=%.2f, Z=%.2f\n', resultados_validos(puntos_fuera_tol, 1),
resultados_validos(puntos_fuera_tol, 2), resultados_validos(puntos_fuera_tol, 3),
resultados_validos(puntos_fuera_tol, 4));
    end
else
    fprintf(fid, 'Todos los puntos están dentro de tolerancia.\n');
end

fclose(fid);
disp(['Informe generado: ' informe_filename]);
%% Paso 6: Visualización de resultados
figure;
scatter3(resultados_validos(:, 2), resultados_validos(:, 3), resultados_validos(:, 4), 'filled');
title('Resultados de Levantamientos Topográficos');
xlabel('X (este)');
ylabel('Y (norte)');
zlabel('Z (cota)');
grid on;
```

1.4 Pruebas del sistema

Se realizarán pruebas de caja blanca y caja negra estas con un enfoque de programación imperativo ya que no hubo una clasificación de clases y esta enfocado a una estructura en torno a funciones o procedimientos que manipulan datos directamente. Se centra en cómo se deben realizar las operaciones paso a paso para lograr el resultado deseado.

Caja blanca: Las pruebas de caja blanca examinan la lógica interna del código y aseguran que todas las rutas posibles sean probadas. El objetivo es cubrirlo tanto como sea posible.



Descripción del grafo de flujo:

El grafo de flujo se compone de nodos y aristas. Los nodos representan puntos de decisión o acciones específicas dentro del código, mientras que las aristas representan las transiciones entre esos nodos. Cada nodo está numerado para facilitar su referencia.

Descripción

(Nodo 1): Inicio
(Nodo 2): Leer archivo
(Nodo 3): Preprocesamiento de datos
(Nodo 4): Bucle sobre datos
(Nodo 5): validar resultados
(Nodo 6): filtrar resultados no validos
(Nodo 7):calcular errores y tolerancia
(Nodo 8): Mostrar errores y puntos fuera de tolerancia
(Nodo 9):generar informes
(Nodo 10): Visualizar resultados
(Nodo 11): Fin

Alternativas de camino

1. Nodo 1 -> Nodo 2 -> Nodo 3 -> Nodo 4 -> Nodo 5 -> Nodo 6 -> Nodo 7 -> Nodo 8 -> Nodo 9 -> Nodo 10 -> Nodo 11 -> Nodo 12 -> Nodo 13 (camino exitoso)

2. Nodo 1 -> Nodo 2 -> Nodo 3 -> Nodo 10 (datos erróneos)

3. Nodo 1 -> Nodo 2 -> Nodo 3 -> Nodo 4 -> Nodo 5 -> Nodo 6 -> Nodo 7-> Nodo 10 (error fuera de tolerancia)

Pruebas por función

-
```
%% Paso 1: Carga de datos crudos
% Carga del archivo exportado
data = readtable('datos_crudos.csv'); % Cambia 'datos_crudos.csv' por el nombre de tu archivo
```

-
```
%% Paso 2: Preprocesamiento de datos crudos
% Renombrar las columnas para facilitar el acceso
```

```
data.Properties.VariableNames = {'Punto', 'Distancia', 'HD', 'VD', 'X_este', 'Y_norte', 'Z_cota',
'Cod'};

% Convertir variables necesarias a números
data.Distancia = str2double(strrep(data.Distancia, ',', '.'));
data.HD = str2double(strrep(data.HD, ',', '.'));
data.VD = str2double(strrep(data.VD, ',', '.'));
data.X_este = str2double(strrep(data.X_este, ',', '.'));
data.Y_norte = str2double(strrep(data.Y_norte, ',', '.'));
data.Z_cota = str2double(strrep(data.Z_cota, ',', '.'));

% Identificar filas inválidas y eliminarlas
filas_invalidas = any(isnan(data{:, {'Distancia', 'HD', 'VD', 'X_este', 'Y_norte', 'Z_cota'}}), 2);
data = data(~filas_invalidas, :);
```

- `%% Paso 3: Cálculos de levantamientos topográficos (Método de Radiación)`

```
% Coordenadas de referencia de la estación
X0 = 0; % Coordenada X de la estación (cambia según punto de referencia)
Y0 = 0; % Coordenada Y de la estación (cambia según punto de referencia)
Z0 = 0; % Coordenada Z de la estación (cambia según punto de referencia)

% Inicializar matriz para resultados
resultados = zeros(height(data), 4); % [Punto, X_calc, Y_calc, Z_calc]

for i = 1:height(data)
    % Coordenadas polares a cartesianas
    Distancia = data.Distancia(i);
    HD = data.HD(i);
    VD = data.VD(i);

    % Convertir ángulos a radianes
    HD_rad = deg2rad(HD);
    VD_rad = deg2rad(VD);

    % Calcular coordenadas en el plano horizontal
    X_calc = X0 + Distancia * cos(VD_rad) * cos(HD_rad);
    Y_calc = Y0 + Distancia * cos(VD_rad) * sin(HD_rad);
    Z_calc = Z0 + Distancia * sin(VD_rad);

    % Almacenar resultados
    resultados(i, :) = [i, X_calc, Y_calc, Z_calc];
end
```

- `%% Paso 4: Interpretación y validación de resultados`

```
% Validación simple: comprobar que las coordenadas no son NaN y están dentro de un rango esperado
valido = all(~isnan(resultados(:, 2:4)), 2) & resultados(:, 2) >= -10000 & resultados(:, 2) <= 10000
& ...
         resultados(:, 3) >= -10000 & resultados(:, 3) <= 10000 & resultados(:, 4) >= -10000 &
resultados(:, 4) <= 10000;

% Filtrar resultados no válidos (opcional)
resultados_validos = resultados(valido, :);

% Calcular errores y tolerancia
errores = sqrt((resultados_validos(:, 2) - data.X_este).^2 + (resultados_validos(:, 3) -
data.Y_norte).^2 + (resultados_validos(:, 4) - data.Z_cota).^2);
tolerancia = 1000; % Define tu propia tolerancia aquí

% Mostrar errores y puntos fuera de tolerancia
disp('Errores:');
disp(errores);
puntos_fuera_tol = errores > tolerancia;
if any(puntos_fuera_tol)
```

```
    warning('Algunos puntos están fuera de tolerancia.');
    disp('Puntos fuera de tolerancia:');
    disp(array2table(resultados_validos(puntos_fuera_tol, :), 'VariableNames', {'Punto', 'X_este',
'Y_norte', 'Z_cota'}));
else
    disp('Todos los puntos están dentro de tolerancia.');
end
```

- %% Paso 5: Generación del informe

```
% Guardar resultados en un archivo .csv
results_table = array2table(resultados_validos, 'VariableNames', {'Punto', 'X_este', 'Y_norte',
'Z_cota'});
writetable(results_table, 'resultados_levantamientos.csv');

% Crear el informe
informe_filename = 'informe_levantamientos.txt';
fid = fopen(informe_filename, 'w');
fprintf(fid, 'Informe de Levantamientos Topográficos\n\n');

fprintf(fid, 'Errores:\n');
for i = 1:length(errores)
    fprintf(fid, 'Punto %d: %.2f\n', resultados_validos(i, 1), errores(i));
end

fprintf(fid, '\n');
if any(puntos_fuera_tol)
    fprintf(fid, 'Algunos puntos están fuera de tolerancia.\n');
    fprintf(fid, 'Puntos fuera de tolerancia:\n');
    for i = 1:size(resultados_validos(puntos_fuera_tol, :), 1)
        fprintf(fid, 'Punto %d: X=%.2f, Y=%.2f, Z=%.2f\n', resultados_validos(puntos_fuera_tol, 1),
resultados_validos(puntos_fuera_tol, 2), resultados_validos(puntos_fuera_tol, 3),
resultados_validos(puntos_fuera_tol, 4));
    end
else
    fprintf(fid, 'Todos los puntos están dentro de tolerancia.\n');
end

fclose(fid);
disp(['Informe generado: ' informe_filename]);
```

- %% Paso 6: Visualización de resultados

```
figure;
scatter3(resultados_validos(:, 2), resultados_validos(:, 3), resultados_validos(:, 4), 'filled');
title('Resultados de Levantamientos Topográficos');
xlabel('X (este)');
ylabel('Y (norte)');
zlabel('Z (cota)');
grid on;

% Mostrar resultados no válidos (opcional)
if any(~valido)
    warning('Algunos puntos no son válidos y se han excluido de los resultados.');
    disp('Puntos no válidos:');
    disp(array2table(resultados(~valido, :), 'VariableNames', {'Punto', 'X_este', 'Y_norte',
'Z_cota'}));
end
```

Caja negra: Las pruebas de caja negra se centran en la funcionalidad del sistema sin considerar su implementación interna enfocándose en su comportamiento de entrada y salida.

```
%% Prueba de caja negra - Equivalencia de Clases (Distancias)
% Descripción: Verificar que las distancias medidas estén dentro de un rango específico.
% Entradas esperadas: Distancias medidas en la columna 'Distancia' de los datos procesados.
% Resultados esperados: Todas las distancias deben estar dentro del rango especificado.

% Definir el rango permitido para las distancias (en metros)
rango_minimo = 0;
rango_maximo = 1000;

% Obtener las distancias medidas
distancias_medidas = data.Distancia;

% Verificar que todas las distancias estén dentro del rango permitido
distancias_validas = all(distancias_medidas >= rango_minimo & distancias_medidas <= rango_maximo);

% Mostrar el resultado de la prueba
if distancias_validas
    disp('Todas las distancias están dentro del rango permitido.');
else
    disp('Algunas distancias están fuera del rango permitido.');
end
```

Buenas prácticas

1.Comentarios descriptivos: Explican claramente el propósito de cada sección del código y las variables utilizadas. Esto mejora la comprensión del código tanto para el autor como para posibles revisores.

Formato del código: Use sangría, espacios en blanco y saltos de línea para mejorar la legibilidad.

Nombres de variables significativos: Los nombres de las variables deben ser descriptivos y reflejar su propósito.

Ejemplo: En el código proporcionado, se utiliza una función llamada calcular coordenadas para encapsular el proceso de cálculo de coordenadas. Esto hace que el código sea más fácil de entender y mantener, ya que la lógica de cálculo está separada del código principal.

2. Eficiencia y rendimiento:

Optimización de bucles: Evite bucles innecesarios y asegúrese de que los cálculos se realicen de manera eficiente.

Preasignación: Asigne memoria a las variables antes de usarlas dentro de bucles para minimizar la sobrecarga de asignación de memoria y mejorar el rendimiento.

Ejemplo: En el código proporcionado, las variables se persignan antes de usarlas dentro de bucles, lo que mejora el rendimiento del código.

3. Precisión y manejo de errores:

Verificación de errores: Implemente la validación de entrada, las verificaciones de existencia de archivos y el manejo de excepciones para garantizar la integridad de los datos y la robustez del programa.

Tipos de datos adecuados: Elija los tipos de datos apropiados para las variables, como *double* para decimales y int para enteros.

Ejemplo: En el código proporcionado, se realizan comprobaciones de validez para garantizar que las coordenadas estén dentro de rangos razonables.

4. Documentación y mantenimiento:

Archivo README: Incluya un archivo README que proporcione una descripción general del código, su propósito, instrucciones de uso y dependencias.

Control de versiones: Use un sistema de control de versiones como Git para realizar un seguimiento de los cambios en el código, permitiendo la colaboración y la reversión a versiones anteriores si es necesario.

Ejemplo: El código proporcionado carece de un archivo README y de comentarios explicativos detallados. Se recomienda agregar estos elementos para mejorar la documentación y el mantenimiento del código.

5. Buenas prácticas adicionales de fuentes externas:

Siga las convenciones de codificación de MATLAB: Adhiérase a las convenciones de codificación recomendadas por MATLAB para la coherencia y la facilidad de colaboración entre programadores de MATLAB.
(https://www.ee.columbia.edu/~marios/matlab/MatlabStyle1p5.pdf)

Adopte patrones de diseño: Utilice patrones de diseño como el patrón Singleton para administrar recursos globales y el patrón Factory para crear objetos de manera eficiente. (https://docs.oracle.com/communications/E79102_01/doc.735/e79086/dsdev_design_patterns.htm)

Hitos

Funcionalidad	**Estado**	**Fecha de implementación**
Importación de observaciones crudas a BBDD	✓	18/06
Exportación a sistema	✓	19/06
Preprocesamiento de datos	✓	19/06
Almacenamiento de datos	✓	22/06
Análisis y procesamiento de los datos	✓	22/06
Interpretación y validación de los datos	✓	23/06
Generar informes y visualización	✓	23/06