https://doi.org/10.17163/abyaups.170.6

Capítulo 6

Diagramas de flujo y algoritmos en ingeniería

Propósito del capítulo

La modelación matemática es fundamental en ingeniería, pero la implementación computacional y la estructuración lógica de procesos son igualmente esenciales. Un ingeniero moderno no solo formula ecuaciones, sino que desarrolla algoritmos, simulaciones y procedimientos automatizados.

Este capítulo introduce la construcción de diagramas de flujo y la escritura de algoritmos en pseudocódigo utilizando herramientas profesionales en LaTeX[3].

Al finalizar este capítulo, el lector será capaz de:

- Representar procesos lógicos mediante diagramas de flujo.
- Utilizar TikZ para construir estructuras de decisión y control.
- Escribir algoritmos en pseudocódigo con formato profesional.
- Documentar procedimientos computacionales de forma reproducible.
- Integrar modelos matemáticos con implementación algorítmica.

6.1. Modelo lógico vs. implementación

Un algoritmo es una secuencia finita de instrucciones bien definidas que resuelven un problema específico. En ingeniería, el flujo lógico precede a la programación.

El diagrama de flujo representa la lógica visualmente, mientras que el pseudocódigo formaliza esa lógica independientemente del lenguaje de programación.

6.2. Importancia en ingeniería eléctrica

En ingeniería eléctrica y electrónica es frecuente:

- Programar algoritmos de control.
- Desarrollar simulaciones numéricas.
- Implementar cálculos iterativos.
- Diseñar lógica de protección.
- Documentar firmware y software embebido.

Un modelo matemático sin representación algorítmica es incompleto en entornos industriales.

6.3. Diagramas de flujo

Los diagramas de flujo permiten representar gráficamente la secuencia de operaciones de un procedimiento lógico o computacional.

6.3.1. Elementos fundamentales

Los elementos más utilizados en la elaboración de diagramas de flujo se resumen en la Tabla 6.1

Tabla 6.1: Símbolos básicos en diagramas de flujo

Elemento	Función
Inicio / Fin	Marca el comienzo o término del proceso
Proceso	Cálculo o instrucción
Decisión	Evaluación lógica (Sí / No)
Entrada / Salida	Lectura o impresión de datos
Flecha	Dirección del flujo

6.3.2. Configuración en el preámbulo

Para construir diagramas de flujo con TikZ se recomienda incluir:

```
\usetikzlibrary{shapes.geometric, arrows.meta, positioning}
```

6.4. Ejemplo práctico: cálculo de potencia trifásica

Se desea:

1. Leer V, I y $\cos\phi$.
2. Calcular $P = \sqrt{3}VI\cos\phi$.
3. Verificar si el factor de potencia es menor que 0.8.
4. Generar advertencia si es necesario.

6.4.1. Diagrama de flujo

El siguiente código permite realizar el diagrama de flujo que se presenta en la Fig. 6.1:

```
\begin{figure}[H]
\centering
\begin{tikzpicture}[
node distance=1.6cm,
every node/.style={font=\small},
startstop/.style={
rectangle, rounded corners,
draw=edgered, fill=softred,
align=center, minimum width=3.2cm
},
process/.style={
rectangle,
draw=edgeblue, fill=softblue,
align=center, minimum width=3.4cm
},
decision/.style={
diamond, aspect=2,
draw=edgeyellow, fill=softyellow,
align=center, inner sep=1pt
},
arrow/.style={->, thick}
]

\node (start) [startstop] {Inicio};
\node (input) [process, below of=start] {Leer $V$, $I$, $\cos\phi$};
\node (calc)  [process, below of=input] {$P=\sqrt{3}VI\cos\phi$};
\node (dec)   [decision, below of=calc, yshift=-0.4cm] {$\cos\phi < 0.8$?};

% Ramas laterales compactas
\node (warn) [process, right=1.2cm of dec] {Mostrar advertencia};
\node (ok)   [process, left=1.2cm of dec]  {Operación normal};

% Nodo de unión y fin
\node (join) [coordinate, below=2cm of dec] {};
```

```
\node (end)  [startstop, below=0.8cm of join] {Fin};

% Flujo principal
\draw[arrow] (start) -- (input);
\draw[arrow] (input) -- (calc);
\draw[arrow] (calc) -- (dec);

% Ramas (Sí/No)
\draw[arrow] (dec.east) -- node[above]{Sí} ++(0.4,0) |- (warn);
\draw[arrow] (dec.west) -- node[above]{No} ++(-0.4,0) |- (ok);

% Unión hacia fin
\draw[arrow] (warn) |- (join);
\draw[arrow] (ok)   |- (join);
\draw[arrow] (join) -- (end);

\end{tikzpicture}
\caption{Diagrama de flujo compacto para evaluación del factor de potencia.}
\label{fig:DiagramaFlujo}
\end{figure}
```

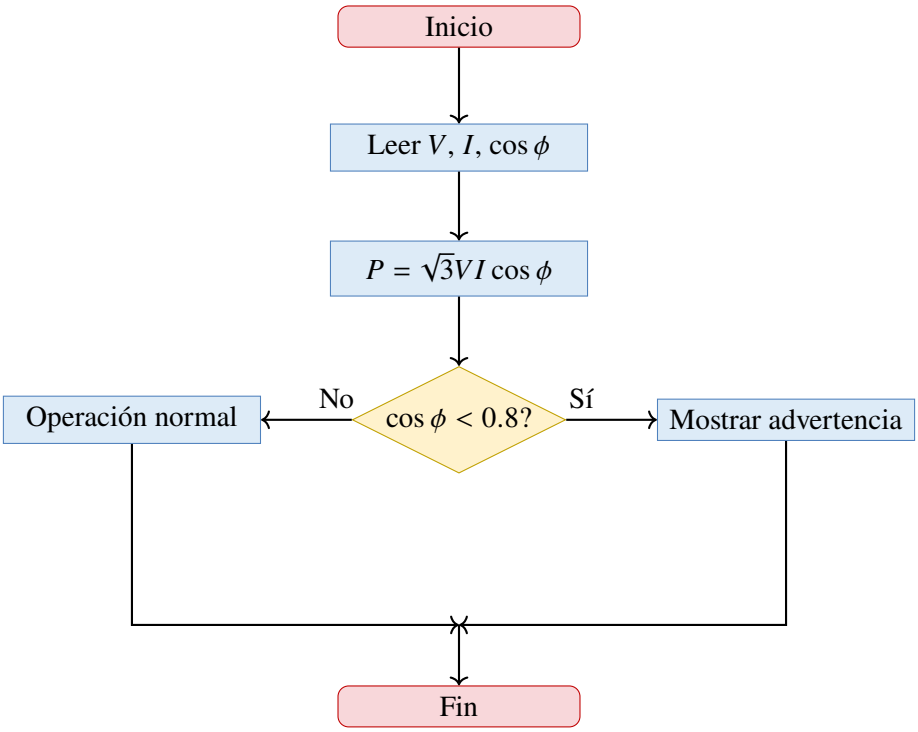

Figura 6.1: Diagrama de flujo compacto para evaluación del factor de potencia.

6.4.2. Comandos y estilos principales para diagramas de flujo (TikZ)

En las Tablas 6.2 y 6.3 se han indicado los principales comando que se utilizan en la elaboración de diagramas de flujo, tales como el que se observa en la Fig. 6.1. Para la definición de colores es necesario incluir en el preámbulo del documento las siguientes instrucciones:

```
\usepackage{xcolor}

\definecolor{softblue}{RGB}{221,235,247}
\definecolor{softyellow}{RGB}{255,242,204}
\definecolor{softred}{RGB}{248,215,218}

\definecolor{edgeblue}{RGB}{79,129,189}
\definecolor{edgeyellow}{RGB}{192,160,0}
\definecolor{edgered}{RGB}{192,0,0}
```

Tabla 6.2: TikZ para diagramas de flujo: librerías, estilos y nodos

Elemento y descripción	Código típico
Cargar librerías: habilita rombos, flechas y posicionamiento relativo	`\usetikzlibrary{shapes.geometric, arrows.meta, positioning}`
Definir estilo Inicio/Fin: caja redondeada para inicio y término	`startstop/.style={rectangle, rounded corners, draw, align=center}`
Definir estilo Proceso: bloque rectangular para cálculos u operaciones	`process/.style={rectangle, draw, align=center}`
Definir estilo Decisión: rombo para condiciones Sí/No	`decision/.style={diamond, aspect=2, draw, align=center}`
Definir estilo de flecha: flechas uniformes y más visibles	`arrow/.style={->, thick}`
Crear un nodo (bloque): genera un bloque con identificador	`\node (A) [process] {Texto};`
Colocar debajo de otro nodo (requiere node distance)	`\node (B) [process, below of=A] {...};`
Colocar a la derecha/izquierda (requiere positioning)	`\node (C) [process, right=1cm of A] {...};`

Tabla 6.3: TikZ para diagramas de flujo: conexiones y rutas

Elemento y descripción	Código típico
Conectar con flecha simple: conexión recta entre nodos	`\draw[arrow] (A) - (B);`
Ruta en L (horizontal y luego vertical): útil para ramificaciones	`\draw[arrow] (A) |- (B);`
Ruta en L invertida (vertical y luego horizontal): reconexión de ramas	`\draw[arrow] (A) -| (B);`
Etiquetar una flecha (Sí/No u otra decisión)	`\draw[arrow] (A) - node[above]{Sí} (B);`
Crear tramo corto antes de doblar (para acortar ramas)	`- ++(0.4,0) |- (B)`
Nodo de unión invisible (recombinación de ramas)	`\node (join) [coordinate] {};`

6.5. Estructuras iterativas

En ingeniería es frecuente utilizar ciclos:

- Método de Newton–Raphson.
- Simulación temporal discreta.
- Integración numérica.

Ejemplo conceptual:
Mientras el error sea mayor que una tolerancia, repetir cálculo.

6.6. Escritura de algoritmos en pseudocódigo

El pseudocódigo permite describir algoritmos sin depender de un lenguaje específico (Python, C, MATLAB, etc.).

6.6.1. Paquetes recomendados

En el preámbulo:

```
\usepackage{algorithm}
\usepackage{algpseudocode}
```

6.6.2. Ejemplo en pseudocódigo

La producción del pseudocódigo que se muestra en El Algoritmo 1, para el diagrama de flujo que se presenta en la Fig. 6.1, se puede obtener mediante los siguientes comandos:

```
\begin{algorithm}
\caption{Cálculo de potencia trifásica}
\begin{algorithmic}[1]
\Require $V$, $I$, $\cos\phi$
\Ensure Potencia $P$

\State $P \gets \sqrt{3} V I \cos\phi$

\If{$\cos\phi < 0.8$}
    \State Mostrar advertencia
\EndIf

\State \Return $P$
\end{algorithmic}
\end{algorithm}
```

Algorithm 1 Cálculo de potencia trifásica

Require: V, I, $\cos\phi$
Ensure: Potencia P
1: $P \leftarrow \sqrt{3} V I \cos\phi$
2: **if** $\cos\phi < 0.8$ **then**
3: Mostrar advertencia
4: **end if**
5: **return** P

6.7. Estructuras básicas en pseudocódigo

En la Tabla 6.4, se resumen las principales instrucciones utilizadas para la generación de pseudocódigos.

Tabla 6.4: Comandos principales en `algpseudocode`

Comando	Función
`\State`	Instrucción simple
`\If ... \EndIf`	Estructura condicional
`\For ... \EndFor`	Ciclo con contador
`\While ... \EndWhile`	Ciclo condicional
`\Require`	Variables de entrada
`\Ensure`	Resultado esperado
`\Return`	Valor de salida

6.8. Ejemplo iterativo: método de Newton–Raphson

En la Fig. 6.2 se representa el diagrama de flujo correspondiente al algoritmo iterativo de Newton Raphson. Para generar este código se utilizaron las isguientes instrucciones:

```
\begin{figure}[H]
\centering
\begin{tikzpicture}[
node distance=1.2cm,
every node/.style={font=\small},
startstop/.style={
rectangle,
rounded corners,
draw=edgeblue,
fill=softblue,
align=center,
minimum width=3.4cm
},
process/.style={
rectangle,
draw=edgeblue,
fill=softblue,
align=center,
minimum width=4.2cm
},
decision/.style={
diamond,
aspect=2.2,
draw=edgeyellow,
fill=softyellow,
align=center,
inner sep=1pt
},
arrow/.style={->, thick}
```

```
]

\node (start) [startstop] {Inicio};
\node (init)  [process, below of=start] {$x \leftarrow x_0$};
\node (eval)  [process, below of=init] {Evaluar $f(x)$ y $f'(x)$};
\node (dec)   [decision, below of=eval, yshift=-0.25cm]
{$|f(x)| > \varepsilon$?};

\node (update)[process, below=0.9cm of dec]
{$x \leftarrow x - \dfrac{f(x)}{f'(x)}$};

\node (end) [startstop, left=1.6cm of dec] {Fin};

% Flujo principal
\draw[arrow] (start) -- (init);
\draw[arrow] (init) -- (eval);
\draw[arrow] (eval) -- (dec);

% Rama NO
\draw[arrow] (dec.west) -- node[above]{No} (end.east);

% Rama SÍ
\draw[arrow] (dec.south) -- node[right]{Sí} (update.north);

% Loop
\draw[arrow] (update.east) -- ++(0.6,0) |- (eval.east);

\end{tikzpicture}
\caption{Diagrama de flujo del método de Newton--Raphson.}
\label{fig:DiaFlujoNewton}
\end{figure}
```

Previamente debe incorporarse en el preámbulo las siguientes definiciones:

```
\usepacage{xcolor}

\definecolor{softblue}{RGB}{221,235,247}
\definecolor{softyellow}{RGB}{255,242,204}
\definecolor{edgeblue}{RGB}{79,129,189}
\definecolor{edgeyellow}{RGB}{192,160,0}
```

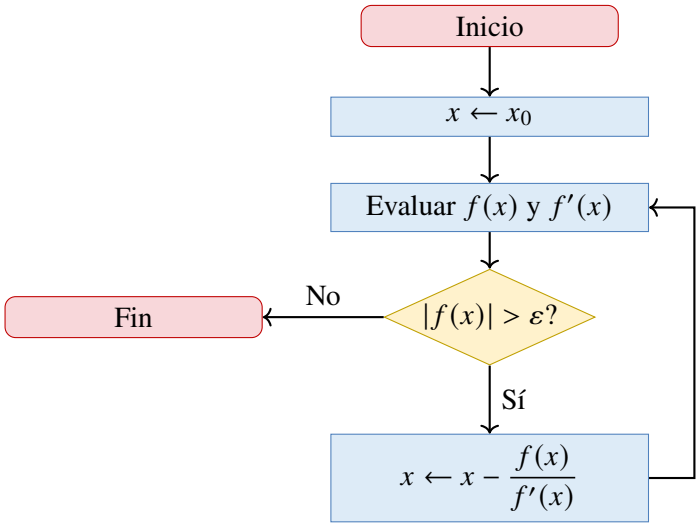

Figura 6.2: Diagrama de flujo del método de Newton–Raphson.

Mediante el código siguiente, se puede obtener el pseudocódigo que se muestra en el Algoritmo 2

```
\begin{algorithm}
\caption{Newton-Raphson\label{alg:NewtonRapshon}}
\begin{algorithmic}[1]
\Require Función $f(x)$, derivada $f'(x)$, valor inicial $x_0$
\Ensure Raíz aproximada $x$

\State $x \gets x_0$

\While{$|f(x)| > \varepsilon$}
    \State $x \gets x - \dfrac{f(x)}{f'(x)}$
\EndWhile

\State \Return $x$
\end{algorithmic}
\end{algorithm}
```

Resultado:

Algorithm 2 Newton–Raphson

Require: Función $f(x)$, derivada $f'(x)$, valor inicial x_0
Ensure: Raíz aproximada x

1: $x \leftarrow x_0$
2: **while** $|f(x)| > \varepsilon$ **do**
3: $\quad x \leftarrow x - \dfrac{f(x)}{f'(x)}$
4: **end while**
5: **return** x

El Algoritmo 2 muestra una estructura iterativa típica. Este tipo de algoritmo es común en análisis de sistemas de potencia y control.

6.9. Comparación: diagrama vs. pseudocódigo

En la Tabla 6.5, se resume una comparación caulitativa para definir el uso de los diagrams de flujo o el psudocódigo en un texto determinado.

Tabla 6.5: Comparación conceptual

Aspecto	Diagrama de flujo	Pseudocódigo
Visual	Sí	Parcial
Formal	Medio	Alto
Uso en tesis	Introductorio	Muy frecuente
Claridad lógica	Alta	Muy alta

Ambos son complementarios y deben emplearse según el contexto.

6.10. Buenas prácticas

- No sobrecargar el diagrama con exceso de texto.
- Mantener coherencia en símbolos.
- Numerar algoritmos si el documento es formal.
- Usar nombres descriptivos de variables.
- Referenciar algoritmos con `\label` y `\ref`.

6.10.1. Flujo típico en proyectos de ingeniería

En entornos industriales y académicos, el proceso suele seguir:

$$\text{Modelo matemático} \rightarrow \text{Diagrama de flujo} \rightarrow \text{Pseudocódigo} \rightarrow \\ \rightarrow \text{Pseudocódigo} \rightarrow \text{Implementación (Python/MATLAB/C)}$$

Esta secuencia garantiza trazabilidad y reproducibilidad.

6.11. Implementación y documentación de código fuente

Una vez definido el modelo matemático, estructurado el diagrama de flujo y formalizado el pseudocódigo, el siguiente paso natural en ingeniería es la implementación computacional.

En documentación técnica profesional es frecuente incluir fragmentos de código fuente en anexos para garantizar reproducibilidad, validación y trazabilidad del trabajo desarrollado.

6.11.1. Inserción de código con resaltado sintáctico

El paquete `listings` permite insertar código fuente con resaltado de sintaxis, numeración de líneas y formato profesional.

En el preámbulo del documento debe incluirse:

```
\usepackage{listings}
\usepackage{xcolor}

\definecolor{softblue}{RGB}{221,235,247}
\definecolor{softyellow}{RGB}{255,242,204}
\definecolor{edgeblue}{RGB}{79,129,189}
\definecolor{edgeyellow}{RGB}{192,160,0}
\definecolor{softred}{RGB}{248,215,218}     % relleno rojo pastel
\definecolor{edgered}{RGB}{192,0,0}         % borde rojo sobrio
\definecolor{codebg}{RGB}{248,248,248}
\definecolor{codeframe}{RGB}{180,180,180}
\definecolor{codekw}{RGB}{0,0,150}
\definecolor{codecom}{RGB}{0,110,0}
\definecolor{codestr}{RGB}{150,0,0}
\definecolor{codenum}{RGB}{90,90,90}

\lstdefinestyle{EngineeringCode}{
  backgroundcolor=\color{codebg},
  frame=single,
  rulecolor=\color{codeframe},
  basicstyle=\ttfamily\small,
  keywordstyle=\color{codekw}\bfseries,
  commentstyle=\color{codecom}\itshape,
  stringstyle=\color{codestr},
  numbers=left,
  numberstyle=\tiny\color{codenum},
  stepnumber=1,
  numbersep=8pt,
  tabsize=2,
  showstringspaces=false,
  breaklines=true,
  breakatwhitespace=true,
  captionpos=b
}
```

Un programa en Python quedaría de esta forma:

```python
import numpy as np

def newton(f, df, x0, tol=1e-6, max_iter=50):
    x = x0
    for k in range(max_iter):
        if abs(f(x)) < tol:
            return x
```

```
        x = x - f(x)/df(x)
    return x

# Ejemplo
f  = lambda x: x**2 - 2
df = lambda x: 2*x
root = newton(f, df, 1.0)
print("Raz aproximada:", root)
```

Listing 6.1: Implementación del método de Newton–Raphson en Python.

Como se muestra en el Listado 6.1, la implementación sigue exactamente la estructura del diagrama de flujo presentado previamente.

Para un listado en Matlab el resultados sería este:

```
V = 13.8e3;       % Voltaje en voltios
I = 120;          % Corriente en amperios
pf = 0.85;        % Factor de potencia

P = sqrt(3)*V*I*pf;

fprintf("Potencia activa = %.2f kW\n", P/1e3);
```

Listing 6.2: Cálculo de potencia trifásica en MATLAB.

La inserción de listados de programas se puede realizar con el siguiente código para un programa en Python:

```
\lstinputlisting[
style=EngineeringCode,
language=Python,
caption={Script completo de simulación.},
label={lst:script_completo}
]{scripts/simulacion.py}
```

Y de esta forma si el código biene de un programa en Matlab:

```
\begin{lstlisting}[style=EngineeringCode,
language=Matlab,
caption={Cálculo de potencia trifásica en MATLAB.},
label={lst:potencia_matlab}]
V = 13.8e3;      % Voltaje en voltios
I = 120;         % Corriente en amperios
pf = 0.85;       % Factor de potencia
P = sqrt(3)*V*I*pf;
fprintf("Potencia activa = %.2f kW\n", P/1e3);
\end{lstlisting}
```

La inclusión del código fuente permite cerrar el ciclo completo del proceso ingenieril:

$$\text{Modelo matemático} \rightarrow \text{Diagrama de flujo} \rightarrow \text{Pseudocódigo} \rightarrow \\ \rightarrow \text{Implementación} \rightarrow \text{Documentación}$$

Este enfoque garantiza reproducibilidad, trazabilidad y rigor técnico.

6.12. Actividad de aplicación

Ejercicio

Diseñar el diagrama de flujo y el pseudocódigo para:

- Simulación discreta de un circuito RLC.
- Control ON/OFF de temperatura.
- Cálculo iterativo de carga en un capacitor.

Trabajo autónomo

Desarrollar un mini-reporte que incluya:

- Un modelo matemático.
- Un diagrama de flujo.
- El pseudocódigo correspondiente.
- Discusión técnica del algoritmo.

Competencias desarrolladas

En este capítulo se han podido desarrollar las siguientes competencias:

- Modelado lógico de procesos de ingeniería.
- Traducción entre diagrama, pseudocódigo e implementación.
- Representación formal de algoritmos iterativos.
- Documentación profesional de código fuente.
- Integración entre análisis matemático y programación.

Cierre del capítulo

La documentación técnica en ingeniería no se limita a ecuaciones y gráficos. Los algoritmos constituyen el puente entre la teoría y la implementación.

Dominar la representación lógica mediante diagramas de flujo y pseudocódigo permite al ingeniero comunicar soluciones reproducibles y estructuradas, competencia esencial en investigación y desarrollo tecnológico.