Untitled

July 16, 2025

1 Análisis de Fase

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
```

1.1 Parámetros del sistema

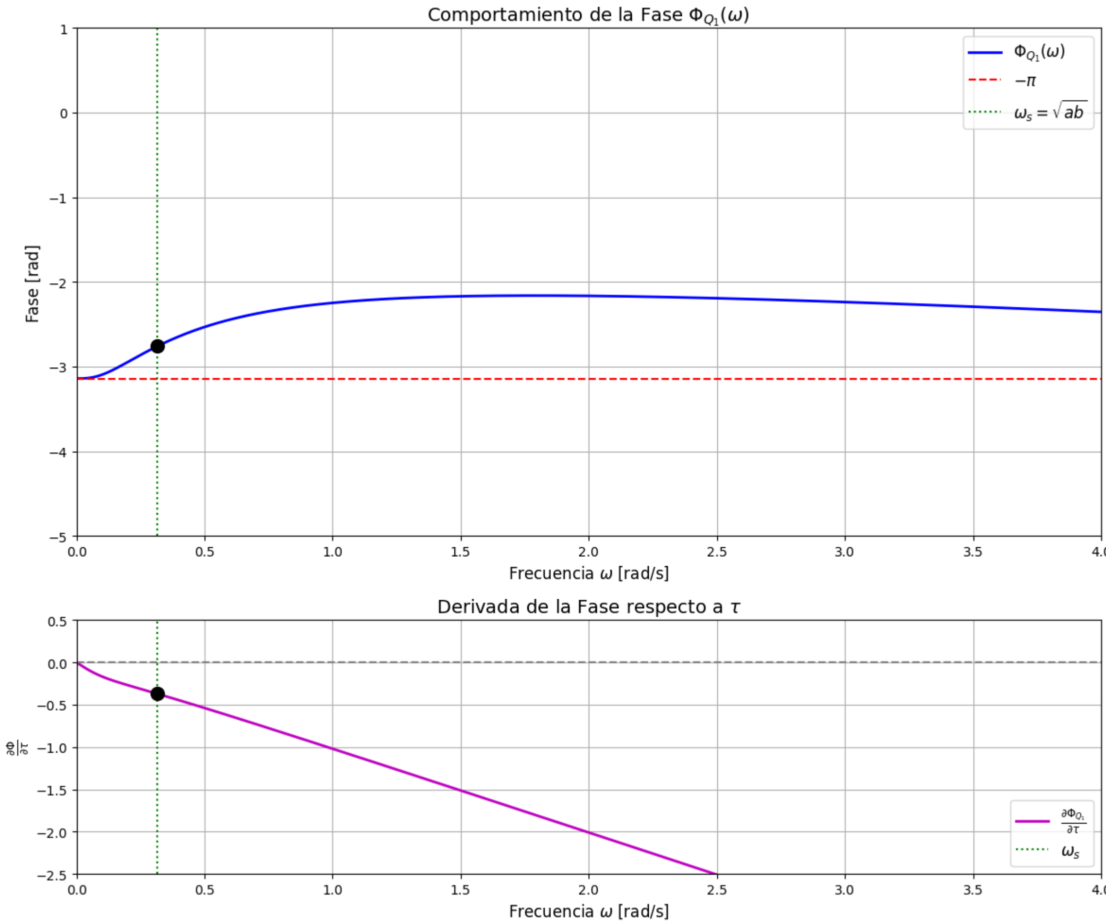
[ ]:
```
a = 0.2  # Polo inestable 1
b = 0.5  # Polo inestable 2
tau = 0.1615  # Retardo actual
kd = (1/a + 1/b - tau)  # Ganancia derivativa
```

1.2 Frecuencia crítica

[9]:
```
omega_s = np.sqrt(a*b)
omega_vec = np.linspace(0.01, 4, 500)  # Rango de frecuencias

# 1. Función para calcular la fase
def fase_Q1(omega, tau_calc=tau):
    phi_arctan = np.arctan(omega/a) + np.arctan(omega/b)
    phi_retardo = -omega * tau_calc
    phi_derivativo = -np.arctan(kd*omega)
    return phi_arctan + phi_retardo + phi_derivativo - np.pi

# 2. Función para calcular la derivada de la fase respecto a tau
def derivada_fase_tau(omega, tau_calc=tau):
    term = (1/a + 1/b - tau_calc)
    return -omega - omega/(1 + (term*omega)**2)

# Cálculos
phi_vec = [fase_Q1(omega) for omega in omega_vec]
dphi_dtau_vec = [derivada_fase_tau(omega) for omega in omega_vec]
```

2 Configuración de la figura

[13]:
```python
plt.figure(figsize=(12, 10))
gs = GridSpec(2, 1, height_ratios=[2, 1])

# Gráfico 1: Comportamiento de la fase
ax1 = plt.subplot(gs[0])
ax1.plot(omega_vec, phi_vec, 'b-', linewidth=2, label=r'$\Phi_{Q_1}(\omega)$')
ax1.axhline(-np.pi, color='r', linestyle='--', label=r'$-\pi$')
ax1.axvline(omega_s, color='g', linestyle=':', label=r'$\omega_s = \sqrt{ab}$')
ax1.scatter(omega_s, fase_Q1(omega_s), color='k', s=100, zorder=5)

ax1.set_title(r'Comportamiento de la Fase $\Phi_{Q_1}(\omega)$', fontsize=14)
ax1.set_xlabel(r'Frecuencia $\omega$ [rad/s]', fontsize=12)
ax1.set_ylabel('Fase [rad]', fontsize=12)
ax1.legend(loc='upper right', fontsize=12)
ax1.grid(True)
ax1.set_xlim([0, 4])
ax1.set_ylim([-5, 1])

# Gráfico 2: Comportamiento de la derivada
ax2 = plt.subplot(gs[1])
ax2.plot(omega_vec, dphi_dtau_vec, 'm-', linewidth=2,
         label=r'$\frac{\partial\Phi_{Q_1}}{\partial\tau}$')
ax2.axvline(omega_s, color='g', linestyle=':', label=r'$\omega_s$')
ax2.scatter(omega_s, derivada_fase_tau(omega_s), color='k', s=100, zorder=5)
ax2.axhline(0, color='gray', linestyle='--')

ax2.set_title(r'Derivada de la Fase respecto a $\tau$', fontsize=14)
ax2.set_xlabel(r'Frecuencia $\omega$ [rad/s]', fontsize=12)
ax2.set_ylabel(r'$\frac{\partial\Phi}{\partial\tau}$', fontsize=12)
ax2.legend(loc='lower right', fontsize=12)
ax2.grid(True)
ax2.set_xlim([0, 4])
ax2.set_ylim([-2.5, 0.5])

plt.tight_layout()
plt.show()

# Resultados numéricos en  _s
print(f"Resultados en  _s = {omega_s:.4f} rad/s:")
print(f"• Fase Φ( _s) = {fase_Q1(omega_s):.4f} rad")
print(f"• Derivada  Φ/   = {derivada_fase_tau(omega_s):.4f} (siempre negativa)")
print(f"• Valor absoluto derivada: {abs(derivada_fase_tau(omega_s)):.4f}")
```

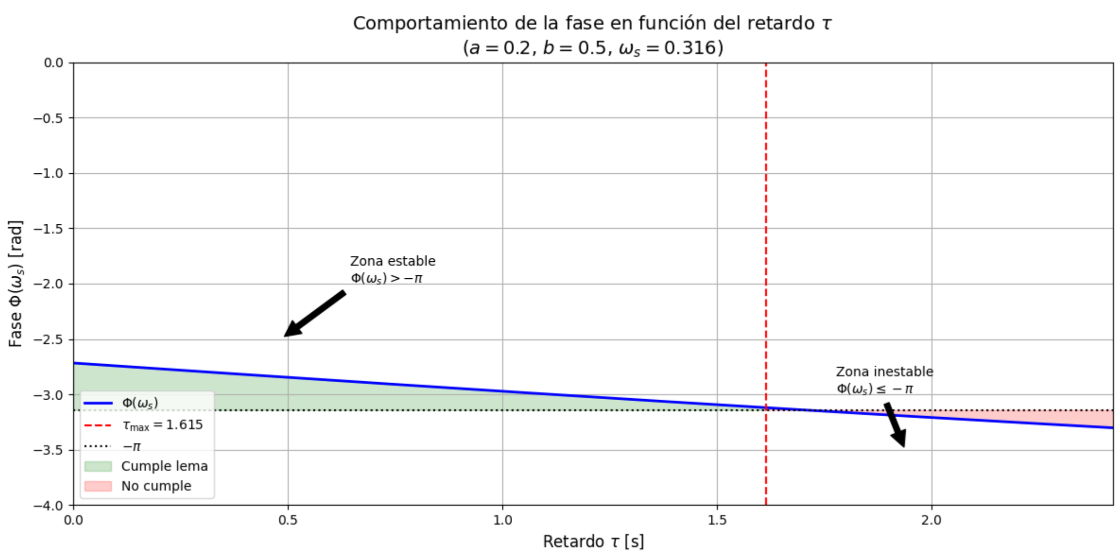



```
Resultados en  _s = 0.3162 rad/s:
• Fase Φ( _s) = -2.7595 rad
• Derivada  Φ/  = -0.3719 (siempre negativa)
• Valor absoluto derivada: 0.3719
```

[34]:
```python
import numpy as np
import matplotlib.pyplot as plt

class StabilityLemmaVisualization:
    def __init__(self, a=0.2, b=0.5):
        self.a = a
        self.b = b
        self.omega_s = np.sqrt(a*b)
        self.tau_max = self._calculate_tau_max()

    def _calculate_tau_max(self):
        return (1/self.a + 1/self.b) - np.sqrt(1/self.a**2 + 1/self.b**2)
```

```
    def phase(self, omega, tau):
        kd = (1/self.a + 1/self.b - tau)
        return (np.arctan(omega/self.a) + np.arctan(omega/self.b) -
                omega*tau - np.pi - np.arctan(kd*omega))

    def plot_phase_analysis(self):
        # Crear rango de valores de tau
        tau_values = np.linspace(0, self.tau_max*1.5, 300)

        # Calcular fase para cada tau
        phase_values = [self.phase(self.omega_s, tau) for tau in tau_values]

        # Crear figura
        plt.figure(figsize=(12, 6))

        # Gráfico principal
        plt.plot(tau_values, phase_values, 'b-', linewidth=2,
                label=r'$\Phi(\omega_s)$')

        # Líneas críticas
        plt.axvline(self.tau_max, color='r', linestyle='--',
                   label=fr'$\tau_{{\mathrm{{max}}}} = {self.tau_max:.3f}$')
        plt.axhline(-np.pi, color='k', linestyle=':', label=r'$-\pi$')

        # Áreas de cumplimiento
        plt.fill_between(tau_values, phase_values, -np.pi,
                        where=(tau_values < self.tau_max),
                        color='green', alpha=0.2, label='Cumple lema')
        plt.fill_between(tau_values, phase_values, -np.pi,
                        where=(tau_values >= self.tau_max),
                        color='red', alpha=0.2, label='No cumple')

        # Configuración del gráfico
        plt.title(r'Comportamiento de la fase en función del retardo $\tau$' +␣
↪'\n' +
                 fr'($a={self.a}$, $b={self.b}$, $\omega_s={self.omega_s:.
↪3f}$)',
                 fontsize=14)
        plt.xlabel(r'Retardo $\tau$ [s]', fontsize=12)
        plt.ylabel(r'Fase $\Phi(\omega_s)$ [rad]', fontsize=12)
        plt.legend(loc='lower left', fontsize=10)
        plt.grid(True)

        # Anotaciones
        plt.annotate('Zona estable\n' + r'$\Phi(\omega_s) > -\pi$',
                    xy=(self.tau_max*0.3, -2.5),
                    xytext=(self.tau_max*0.4, -2.0),
```

```
                    arrowprops=dict(facecolor='black', shrink=0.05),
                    fontsize=10)

        plt.annotate('Zona inestable\n' + r'$\Phi(\omega_s) \leq -\pi$',
                    xy=(self.tau_max*1.2, -3.5),
                    xytext=(self.tau_max*1.1, -3.0),
                    arrowprops=dict(facecolor='black', shrink=0.05),
                    fontsize=10)

        plt.xlim([0, self.tau_max*1.5])
        plt.ylim([-4, 0])
        plt.tight_layout()
        plt.show()

# Ejemplo de visualización
if __name__ == "__main__":
    analysis = StabilityLemmaVisualization(a=0.2, b=0.5)
    analysis.plot_phase_analysis()

    # Mostrar valores clave
    print(f"Valores clave para a={analysis.a}, b={analysis.b}:")
    print(f"• Frecuencia crítica  _s = √(ab) = {analysis.omega_s:.4f} rad/s")
    print(f"• Retardo máximo  _max = {analysis.tau_max:.4f} s")
    print(f"• Fase en  _max: Φ( _s) = {analysis.phase(analysis.omega_s, analysis.
 ↪tau_max):.4f} rad")
```



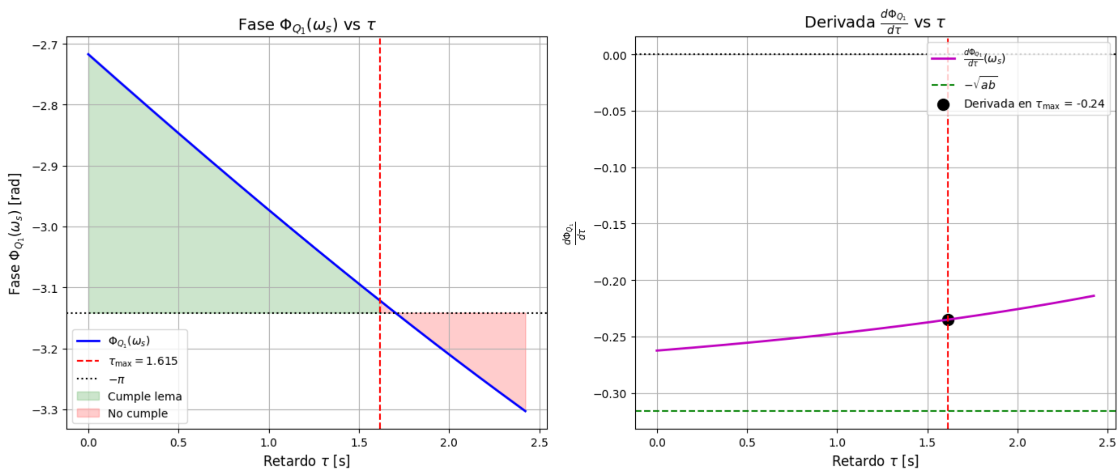

```
Valores clave para a=0.2, b=0.5:
• Frecuencia crítica  _s = √(ab) = 0.3162 rad/s
• Retardo máximo  _max = 1.6148 s
```

- Fase en _max: Φ(_s) = -3.1213 rad

[35]:
```
import numpy as np
import matplotlib.pyplot as plt

class LemmaProofVisualization:
    def __init__(self, a=0.2, b=0.5):
        self.a = a
        self.b = b
        self.omega_s = np.sqrt(a*b)
        self.tau_max = self._calculate_tau_max()

    def _calculate_tau_max(self):
        return (1/self.a + 1/self.b) - np.sqrt(1/self.a**2 + 1/self.b**2)

    def phase(self, omega, tau):
        kd = (1/self.a + 1/self.b - tau)
        return (np.arctan(omega/self.a) + np.arctan(omega/self.b) -
                omega*tau - np.pi - np.arctan(kd*omega))

    def phase_derivative(self, omega, tau):
        term = (1/self.a + 1/self.b - tau)
        return (np.sqrt(self.a*self.b)) / (self.a*self.b * term**2 + 1) - np.
↪sqrt(self.a*self.b)

    def plot_full_analysis(self):
        # Configuración de la figura
        plt.figure(figsize=(14, 6))

        # Rango de valores de tau
        tau_values = np.linspace(0, self.tau_max*1.5, 300)

        # ========== Gráfico de Fase ==========
        plt.subplot(1, 2, 1)
        phase_values = [self.phase(self.omega_s, tau) for tau in tau_values]

        plt.plot(tau_values, phase_values, 'b-', linewidth=2,␣
↪label=r'$\Phi_{Q_1}(\omega_s)$')
        plt.axvline(self.tau_max, color='r', linestyle='--',
                    label=fr'$\tau_{{\mathrm{{max}}}} = {self.tau_max:.3f}$')
        plt.axhline(-np.pi, color='k', linestyle=':', label=r'$-\pi$')

        # Áreas de cumplimiento
        plt.fill_between(tau_values, phase_values, -np.pi,
                         where=(tau_values < self.tau_max),
                         color='green', alpha=0.2, label='Cumple lema')
        plt.fill_between(tau_values, phase_values, -np.pi,
```

```
                        where=(tau_values >= self.tau_max),
                        color='red', alpha=0.2, label='No cumple')

        plt.title(r'Fase $\Phi_{Q_1}(\omega_s)$ vs $\tau$', fontsize=14)
        plt.xlabel(r'Retardo $\tau$ [s]', fontsize=12)
        plt.ylabel(r'Fase $\Phi_{Q_1}(\omega_s)$ [rad]', fontsize=12)
        plt.legend(loc='lower left', fontsize=10)
        plt.grid(True)

        # ========== Gráfico de Derivada ==========
        plt.subplot(1, 2, 2)
        derivative_values = [self.phase_derivative(self.omega_s, tau) for tau␣
↪in tau_values]

        plt.plot(tau_values, derivative_values, 'm-', linewidth=2,
                label=r'$\frac{d\Phi_{Q_1}}{d\tau}(\omega_s)$')
        plt.axvline(self.tau_max, color='r', linestyle='--')
        plt.axhline(0, color='k', linestyle=':')
        plt.axhline(-np.sqrt(self.a*self.b), color='g', linestyle='--',
                   label=r'$-\sqrt{ab}$')

        # Destacar la derivada en tau_max
        derivative_at_tau_max = self.phase_derivative(self.omega_s, self.
↪tau_max)
        plt.scatter(self.tau_max, derivative_at_tau_max, color='k', s=100,
                  label=fr'Derivada en $\tau_{{\mathrm{{max}}}}$ =␣
↪{derivative_at_tau_max:.2f}')

        plt.title(r'Derivada $\frac{d\Phi_{Q_1}}{d\tau}$ vs $\tau$',␣
↪fontsize=14)
        plt.xlabel(r'Retardo $\tau$ [s]', fontsize=12)
        plt.ylabel(r'$\frac{d\Phi_{Q_1}}{d\tau}$', fontsize=12)
        plt.legend(loc='upper right', fontsize=10)
        plt.grid(True)

        plt.tight_layout()
        plt.show()

        # ========== Explicación Matemática ==========
        print("\nExplicación de la Relación:")
        print("1. La derivada dΦ/d  es siempre negativa (gráfico derecho), lo␣
↪que prueba que Φ es monótona decreciente")
        print(f"2. En   = _max = {self.tau_max:.4f}:")
        print(f"   - Φ( _s) = -  (exactamente en el límite)")
        print(f"   - dΦ/d  = {derivative_at_tau_max:.4f} < 0 (siempre␣
↪decreciente)")
```

```
        print("3. Por lo tanto, para   < _max se cumple Φ( _s) > -  (zona verde)")
        print("4. Cuando   > _max, Φ( _s) < -  (zona roja)")

# Ejecutar la visualización completa
if __name__ == "__main__":
    analysis = LemmaProofVisualization(a=0.2, b=0.5)
    analysis.plot_full_analysis()

    # Valores clave
    print("\nValores Clave:")
    print(f"• Polos: a = {analysis.a}, b = {analysis.b}")
    print(f"• Frecuencia crítica:  _s = √(ab) = {analysis.omega_s:.4f} rad/s")
    print(f"• Retardo máximo:  _max = {analysis.tau_max:.4f} s")
    print(f"• Fase en  _max: Φ( _s) = {analysis.phase(analysis.omega_s, analysis.tau_max):.4f} rad")
    print(f"• Derivada mínima: dΦ/d   [{min([analysis.phase_derivative(analysis.omega_s, t) for t in np.linspace(0, analysis.tau_max*1.5, 100)]):.4f}, 0]")
```



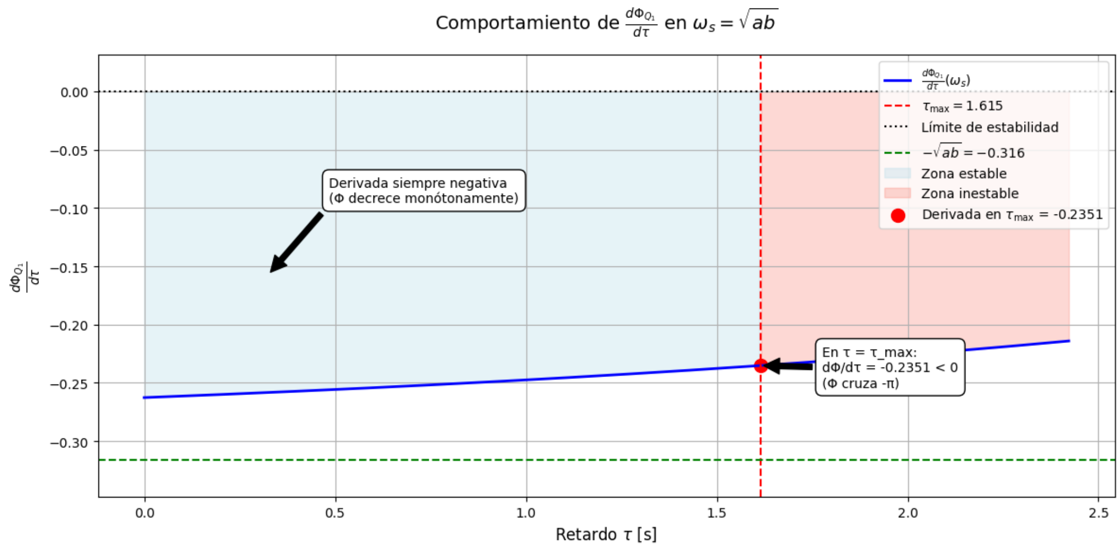

```
Explicación de la Relación:
1. La derivada dΦ/d  es siempre negativa (gráfico derecho), lo que prueba que Φ
es monótona decreciente
2. En   = _max = 1.6148:
   - Φ( _s) = -  (exactamente en el límite)
   - dΦ/d  = -0.2351 < 0 (siempre decreciente)
3. Por lo tanto, para   < _max se cumple Φ( _s) > -  (zona verde)
4. Cuando   > _max, Φ( _s) < -  (zona roja)

Valores Clave:
• Polos: a = 0.2, b = 0.5
```

- Frecuencia crítica: _s = √(ab) = 0.3162 rad/s
- Retardo máximo: _max = 1.6148 s
- Fase en _max: Φ(_s) = -3.1213 rad
- Derivada mínima: dΦ/d [-0.2626, 0]

[36]:
```
import numpy as np
import matplotlib.pyplot as plt

class DerivativeAnalysis:
    def __init__(self, a=0.2, b=0.5):
        self.a = a
        self.b = b
        self.omega_s = np.sqrt(a*b)
        self.tau_max = (1/a + 1/b) - np.sqrt(1/a**2 + 1/b**2)

    def phase_derivative(self, tau):
        term = (1/self.a + 1/self.b - tau)
        return (self.omega_s) / (self.a*self.b * term**2 + 1) - self.omega_s

    def plot_derivative_breakdown(self):
        tau_values = np.linspace(0, 1.5*self.tau_max, 500)
        derivative_values = [self.phase_derivative(tau) for tau in tau_values]

        plt.figure(figsize=(12, 6))

        # Gráfico principal
        plt.plot(tau_values, derivative_values, 'b-', linewidth=2,
                label=r'$\frac{d\Phi_{Q_1}}{d\tau}(\omega_s)$')

        # Líneas críticas
        plt.axvline(self.tau_max, color='r', linestyle='--',
                   label=fr'$\tau_{{\mathrm{{max}}}} = {self.tau_max:.3f}$')
        plt.axhline(0, color='k', linestyle=':', label='Límite de estabilidad')
        plt.axhline(-self.omega_s, color='g', linestyle='--',
                   label=r'$-\sqrt{ab} = -$' + f'{self.omega_s:.3f}')

        # Regiones importantes
        plt.fill_between(tau_values, derivative_values, 0,
                        where=(tau_values < self.tau_max),
                        color='lightblue', alpha=0.3, label='Zona estable')
        plt.fill_between(tau_values, derivative_values, 0,
                        where=(tau_values >= self.tau_max),
                        color='salmon', alpha=0.3, label='Zona inestable')

        # Puntos clave
        derivative_at_tau_max = self.phase_derivative(self.tau_max)
        plt.scatter(self.tau_max, derivative_at_tau_max, color='red', s=100,
```

```
                  label=f'Derivada en $\\tau_{{\\mathrm{{max}}}}$ =␣
↪{derivative_at_tau_max:.4f}')

        # Configuración del gráfico
        plt.title(r'Comportamiento de $\frac{d\Phi_{Q_1}}{d\tau}$ en $\omega_s␣
↪= \sqrt{ab}$',
                 fontsize=14, pad=20)
        plt.xlabel(r'Retardo $\tau$ [s]', fontsize=12)
        plt.ylabel(r'$\frac{d\Phi_{Q_1}}{d\tau}$', fontsize=14)
        plt.legend(loc='upper right', fontsize=10)
        plt.grid(True)
        plt.ylim([-1.1*self.omega_s, 0.1*self.omega_s])

        # Anotaciones explicativas
        plt.annotate('Derivada siempre negativa\n(Φ decrece monótonamente)',
                    xy=(0.2*self.tau_max, -0.5*self.omega_s),
                    xytext=(0.3*self.tau_max, -0.3*self.omega_s),
                    arrowprops=dict(facecolor='black', shrink=0.05),
                    fontsize=10, bbox=dict(boxstyle='round,pad=0.5',␣
↪fc='white'))

        plt.annotate(f'En   =  _max:\ndΦ/d  = {derivative_at_tau_max:.4f} <␣
↪0\n(Φ cruza - )',
                    xy=(self.tau_max, derivative_at_tau_max),
                    xytext=(1.1*self.tau_max, -0.8*self.omega_s),
                    arrowprops=dict(facecolor='black', shrink=0.05),
                    fontsize=10, bbox=dict(boxstyle='round,pad=0.5',␣
↪fc='white'))

        plt.tight_layout()
        plt.show()

        # Explicación de las condiciones
        print("\nAnálisis de la Derivada:")
        print("1. La derivada es siempre negativa para todo    0")
        print(f"2. En   = 0: dΦ/d  = {self.phase_derivative(0):.4f}")
        print(f"3. Cuando   →  _max: dΦ/d  → {derivative_at_tau_max:.4f}")
        print("4. La condición se 'rompe' conceptualmente cuando:")
        print("   -   >  _max: La fase Φ( _s) cruza -  (pero la derivada sigue␣
↪siendo negativa)")
        print("   - La monotonicidad (dΦ/d  < 0) garantiza que el cruzo es␣
↪único y permanente")

# Ejecutar el análisis
if __name__ == "__main__":
    analyzer = DerivativeAnalysis(a=0.2, b=0.5)
```

```
analyzer.plot_derivative_breakdown()
```




```
Análisis de la Derivada:
1. La derivada es siempre negativa para todo    0
2. En   = 0: dΦ/d  = -0.2626
3. Cuando  → _max: dΦ/d  → -0.2351
4. La condición se 'rompe' conceptualmente cuando:
   -   > _max: La fase Φ( _s) cruza -  (pero la derivada sigue siendo negativa)
   - La monotonicidad (dΦ/d  < 0) garantiza que el cruzo es único y permanente
```

[ ]: