Universidad de Buenos Aires
Facultad de Ingeniería

Especialización en Microelectrónica

Dispositivos Semiconductores

Transistor Bipolar de Juntura

Alumno: Agustín Galdeman
Fecha: 28 de mayo de 2025

Buenos Aires, Argentina

Introducción

El objetivo de este trabajo es caracterizar eléctricamente un transistor bipolar de juntura (TBJ) NPN BC547C utilizando simulaciones en LTspice y posterior análisis con Python. Se busca obtener las curvas características del dispositivo, estimar parámetros físicos del modelo ideal y evaluar su comportamiento en pequeña señal.

Parte 1 – Curvas de Transferencia

Se simularon las corrientes I_B e I_C para distintas tensiones V_{BE} manteniendo $V_{CE} = 3$. Los datos fueron compensados considerando I_{CB0} para reflejar únicamente el comportamiento en modo activo directo (MAD).

A partir de estos datos, se graficaron las curvas $I_C - I_{CB0}$ e $I_B + I_{CB0}$ en escala semilogarítmica:

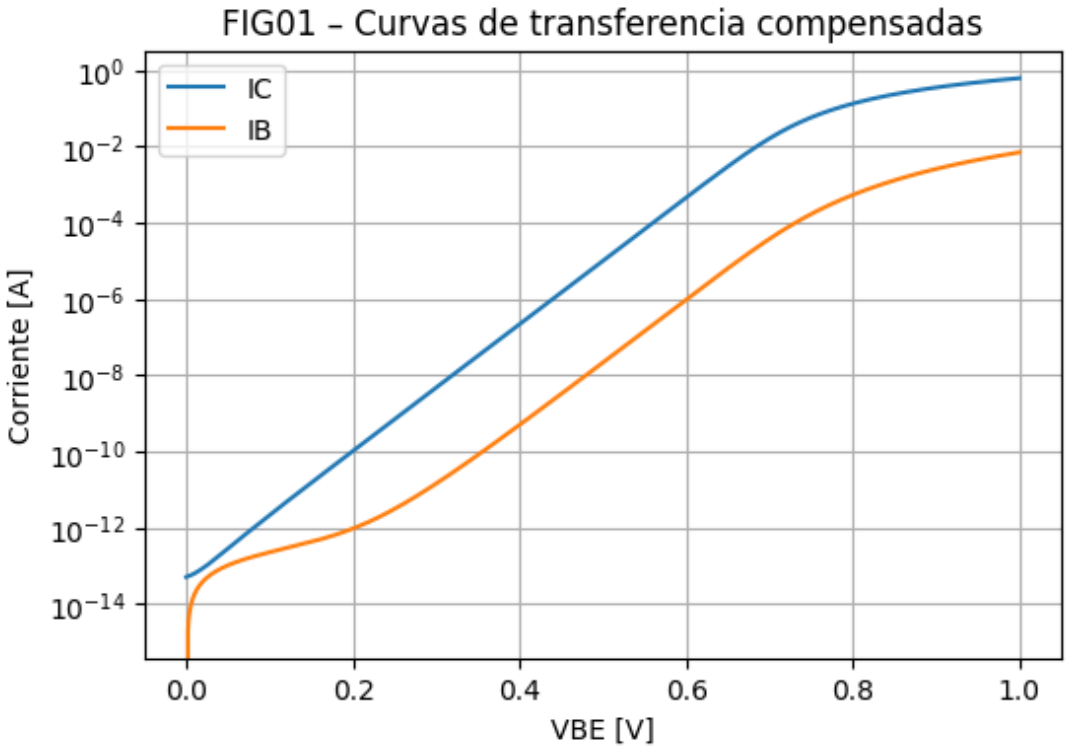


Figura 1: Curvas de transferencia compensadas del TBJ BC547C.

Estimación del Dopaje de la Base

El análisis de las curvas de transferencia permite estimar el dopaje de la base N_A del transistor a partir de las corrientes de saturación extraídas en bajo y alto nivel de inyección.

En condiciones ideales, la corriente de colector en modo activo directo está relacionada con la concentración de dopaje de la base según la siguiente expresión:

$$I_{C0} \propto \frac{D_n \cdot n_i^2}{L_n \cdot N_A} \tag{1}$$

donde D_n es la difusividad de electrones, n_i es la concentración intrínseca de portadores, L_n la longitud de difusión, y N_A el dopaje de la base.

Si se comparan los valores de I_{C0} obtenidos a partir de los ajustes en bajo nivel de inyección (I_{C0}^{BNI}) y en alto nivel de inyección (I_{C0}^{ANI}), se puede estimar la relación entre ellos como:

$$\frac{I_{C0}^{BNI}}{I_{C0}^{ANI}} \approx \frac{N_A^{ANI}}{N_A^{BNI}} \tag{2}$$

Bajo el supuesto de que todos los demás parámetros se mantienen constantes, esta relación permite obtener una estimación relativa del dopaje de la base. Sin embargo, en la práctica, la aparición de efectos secundarios como recombinación adicional, resistencia de base o incluso modulación de la carga almacenada pueden invalidar esta aproximación.

$$N_{aB} = \frac{IC0^{ANI}}{IC0^{BNI}} \cdot \frac{ni}{2} \tag{3}$$

En caso de obtener un valor ilógico (por ejemplo, $N_A < 10^{15}\ cm^{-3}$ o $N_A > 10^{18}\ cm^{-3}$), se deben revisar las siguientes posibles fallas del modelo:

- El transistor opera fuera de la región ideal en alguno de los rangos de V_{BE} utilizados.
- La resistencia de base r_b no fue considerada en el modelo, y afecta la linealidad de la curva.
- Hay una contribución significativa de recombinación no ideal en la base.
- Se producen efectos de auto-calentamiento o variaciones de temperatura locales.

Por tanto, este análisis permite no sólo estimar un parámetro físico del dispositivo, sino también evaluar la validez del modelo ideal en distintas condiciones de operación. El valor obtenido es el siguiente:

$$N_{aB} = 1{,}548 \cdot 10^{20} cm^{-3} \tag{4}$$

Que parece razonable, aunque es un poco alto.

Ajustes lineales para curvas de correinte:

Se realizó un ajuste lineal de la forma:

$$\ln(I) = a \cdot V_{BE} + b \tag{5}$$

para distintos rangos de V_{BE}, en función del régimen dominante, con el objetivo de extraer parámetros como $I_{Br,0}$, I_{B0} e I_{C0} en distintas condiciones de inyección:

- **Región de baja V_{BE}:** donde domina la recombinación en la SCR, se ajustó I_B con pendiente $1/(2V_{th})$.
- **Región de media V_{BE}:** donde domina la inyección, se extrajo I_{B0} y I_{C0}^{BNI} con pendiente $1/V_{th}$.
- **Región de alta V_{BE}:** se obtuvo I_{C0}^{ANI}, donde los efectos de alto nivel de inyección se manifiestan con pendiente $1/(2V_{th})$.

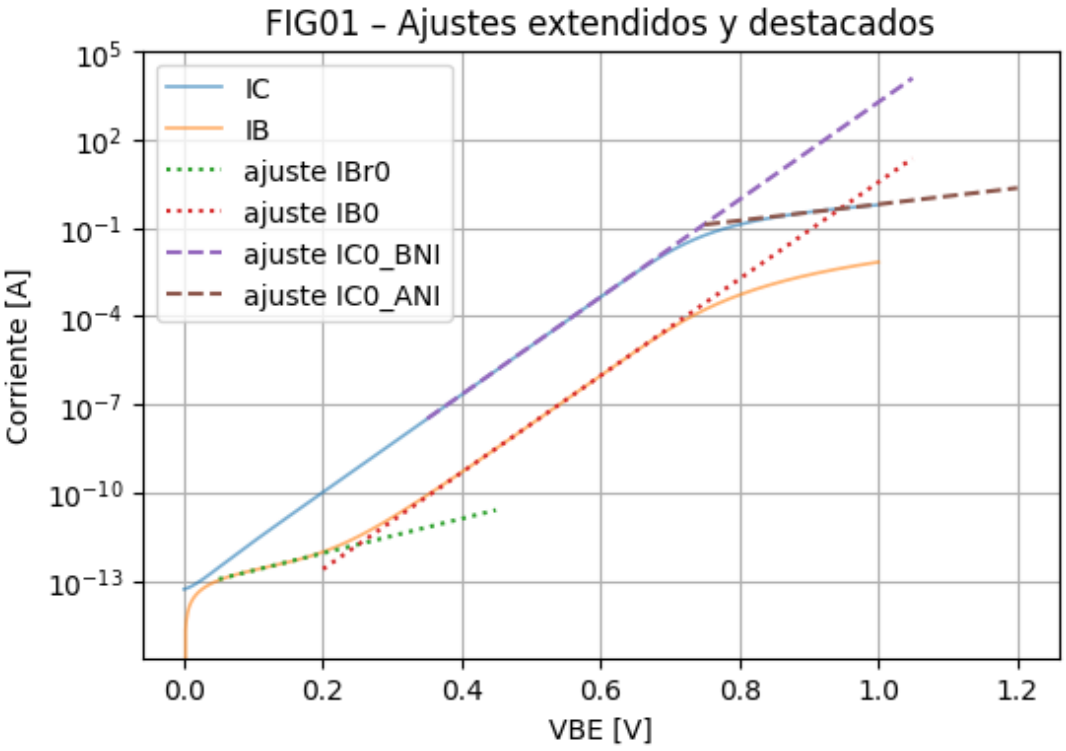


Figura 2: Curvas de transferencia con aproximación lineal del TBJ BC547C.

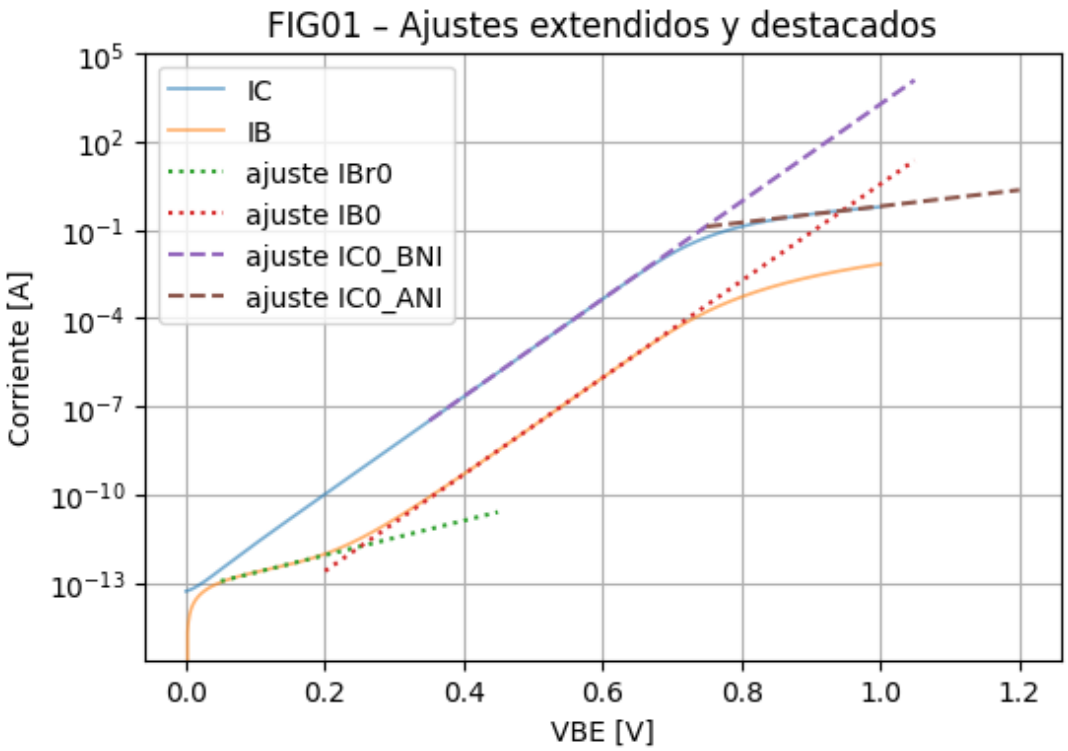


Figura 3: Curvas de transferencia con aproximación lineal del TBJ BC547C.

```
LOW (IBr,0):  a = 1.356e+01  =>  Vth = 7.377e-02 V
MID_IB (IB0):  a = 3.789e+01  =>  Vth = 2.639e-02 V
MID_IC (IC0_BNI):  a = 3.816e+01  =>  Vth = 2.621e-02 V
HIGH (IC0_ANI):  a = 6.368e+00  =>  Vth = 1.570e-01 V
IBr0        Vth = 36.89 mV  FUERA de ±10 %
IB0         Vth = 26.39 mV  OK, Valor dentro de rango
IC0_BNI     Vth = 26.21 mV  OK, Valor dentro de rango
IC0_ANI     Vth = 78.52 mV  FUERA de ±10 %

=== PUNTO 3 - Corrientes características estimadas ===

I_Br,0   = 5.792e-14 A
I_B0     = 1.283e-16 A
I_C0     = 5.186e-14 A   (modo BNI)
I_C0'    = 1.107e-03 A   (modo ANI)
```

Figura 4: Resultados aproximaciones.

Se verificó que el valor de V_{th} estimado en cada ajuste sea consistente con 25.9, valor esperado a temperatura ambiente (27°C). Se encontró concordancia dentro de un 10 % en los casos correspondientes a inyección a bajo nivel, mientras que en regiones extremas el error aumenta, atribuible a la resistencia base r_b ignorada en el modelo.

Cálculo del parámetro β_0

El parámetro β_0 representa la ganancia de corriente en continua del transistor en modo activo directo, definida como la relación entre la corriente de colector y la corriente de base corregidas por la corriente de saturación inversa I_{CB0}:

$$\beta_0 = \frac{I_C - I_{CB0}}{I_B + I_{CB0}} \tag{6}$$

Este parámetro es esencial para el diseño de etapas amplificadoras, ya que determina la eficiencia del transistor en transferir señal desde la base hacia el colector. En general, se espera que β_0 aumente con V_{BE} hasta cierto punto, y luego comience a decrecer debido a efectos como la recombinación en la base y la resistencia base.

A continuación se grafica β_0 en función de I_C en escala logarítmica en ambos ejes para visualizar mejor la evolución del parámetro en varios órdenes de magnitud de corriente:

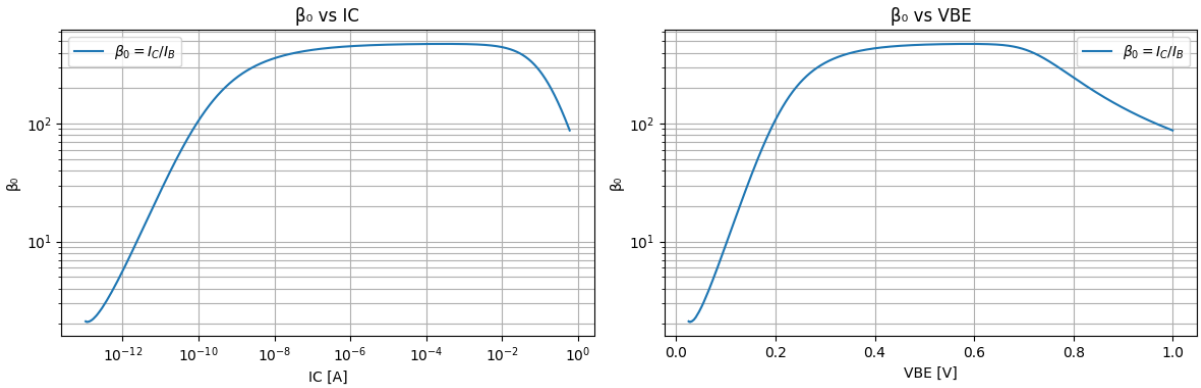


Figura 5: Evolución del parámetro β_0 en función de I_C (escala log-log) y V_{BE}.

Se observa una región de máximo donde el transistor opera con mayor eficiencia. Hacia corrientes muy bajas o muy altas, β_0 tiende a disminuir por los efectos no ideales del modelo físico del TBJ.

Cálculo de la Transconductancia g_m

La transconductancia g_m cuantifica cuánto varía la corriente de colector ante pequeños cambios en la tensión base-emisor V_{BE}, manteniendo V_{CE} constante. Su expresión teórica proviene de la derivada de la ecuación exponencial del modelo MAD:

$$g_m = \frac{\partial I_C}{\partial V_{BE}} \approx \frac{\Delta I_C}{\Delta V_{BE}} \approx \frac{I_C}{V_{th}} \tag{7}$$

Donde $V_{th} \approx 25{,}9$ a temperatura ambiente. Este parámetro es fundamental para evaluar la ganancia de tensión en el modelo de pequeña señal.

A continuación se presentan tres gráficos:

1. g_m en función de I_C en escala lineal.
2. g_m en función de I_C en escala logarítmica en ambos ejes.
3. La relación g_m/I_C en función de I_C, que idealmente debería permanecer constante e igual a $1/V_{th}$.

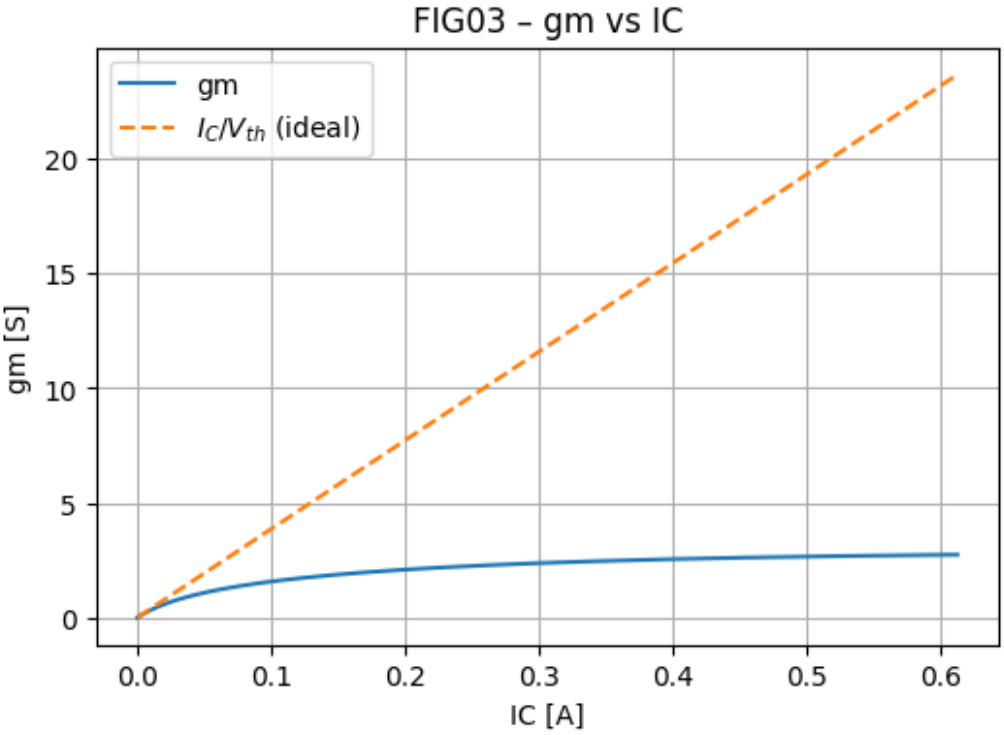


Figura 6: Transconductancia g_m en función de I_C (escala lineal).

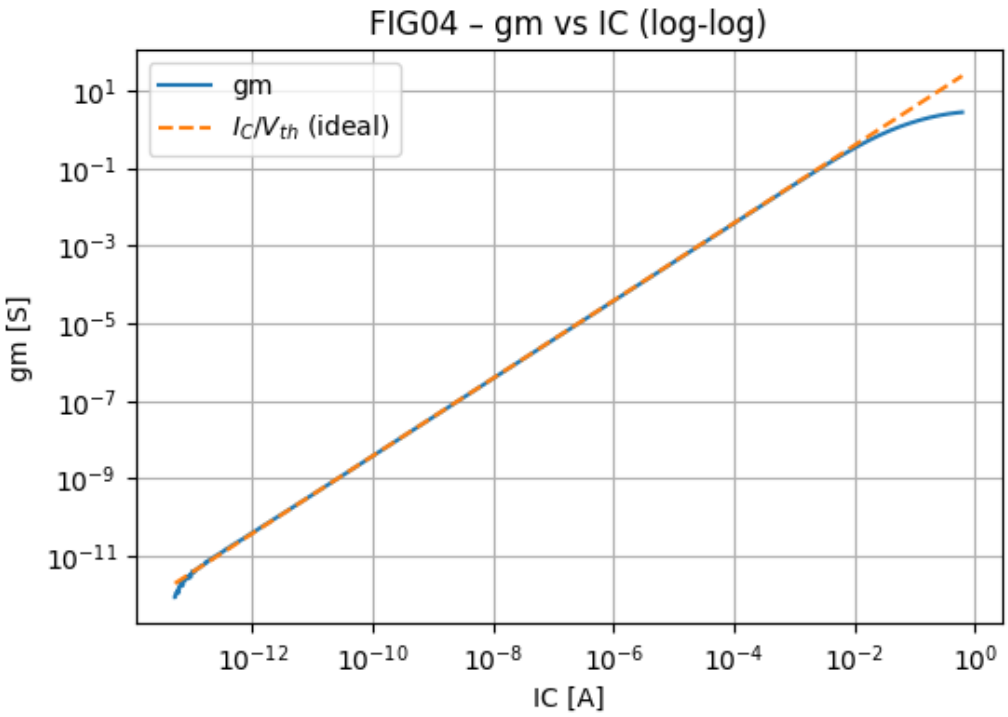


Figura 7: Transconductancia g_m en función de I_C (escala log-log).

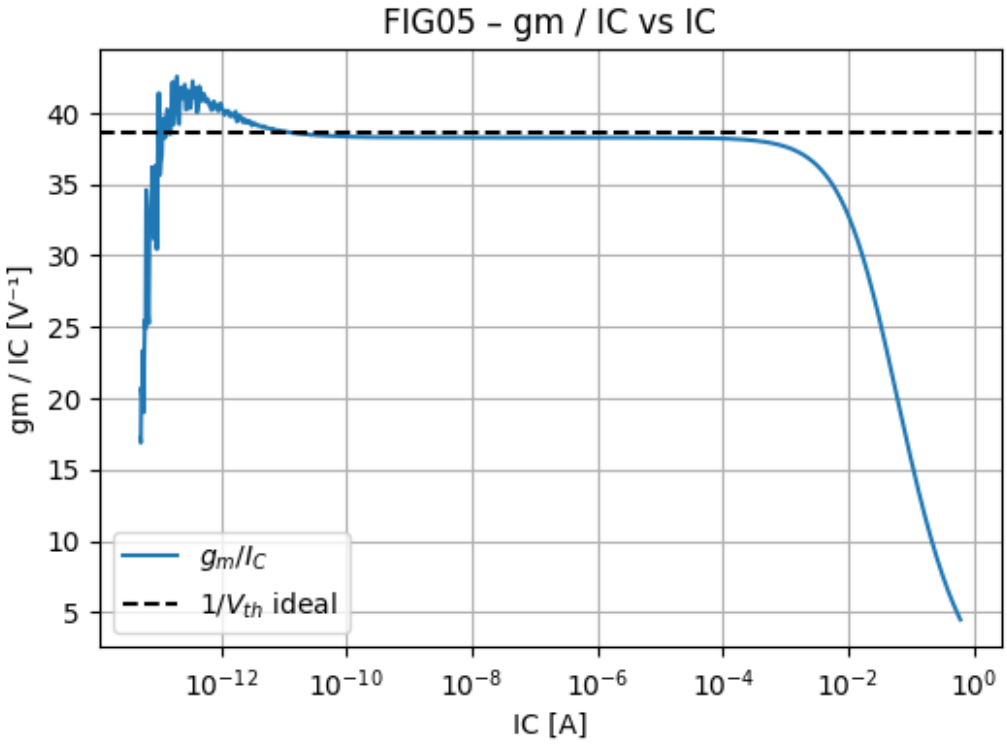


Figura 8: Cociente g_m/I_C vs. I_C. El valor debería mantenerse constante si V_{th} se conserva.

Los resultados obtenidos son consistentes con el modelo ideal para un amplio rango de corrientes. La dispersión en extremos puede atribuirse a errores numéricos, efectos resistivos y discontinuidades en los datos.

Curvas de Salida

Se analizaron las curvas de salida del transistor BC547C simulando I_C en función de V_{CE} para tres valores de polarización de base: $V_{BE} = 620mV$, $665mV$ y $685mV$. Estas simulaciones permiten estudiar el comportamiento del dispositivo en la región activa del modo activo directo (MAD), y obtener parámetros importantes como el voltaje de Early (V_A) y la resistencia de salida (r_o).

1 – Curvas de Salida Simuladas

El circuito de simulación se mantuvo con una fuente de tensión V_{BE} fija y se barrió V_{CE} desde 0 hasta 3. A continuación se presentan las curvas de I_C obtenidas para cada valor de V_{BE}:

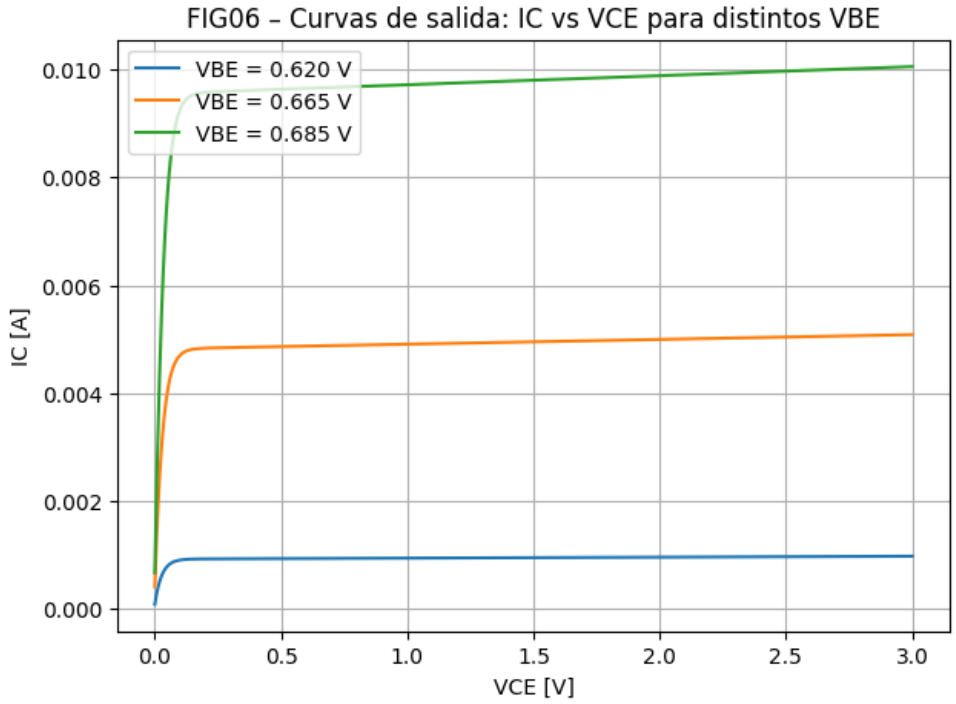


Figura 9: Curvas de salida del TBJ BC547C para distintos valores de V_{BE}.

Las curvas muestran la saturación inicial (bajo V_{CE}), seguida de una región casi lineal donde se manifiesta el efecto Early, con pendiente creciente que indica dependencia de I_C con V_{CE}.

Cálculo del Voltaje de Early y Resistencia de Salida

A partir de las regiones lineales de las curvas se realizó un ajuste de la forma:

$$I_C = I_C(V_{CE} = 0)\left(1 + \frac{V_{CE}}{V_A}\right) \tag{8}$$

De este ajuste se obtuvieron los parámetros:

$$V_A = \frac{I_C(VCE = 0)}{(\partial I_C/\partial V_{CE})} = \frac{b}{a} \tag{9}$$

$$r_o = \frac{V_A + V_{CE}}{I_C} \tag{10}$$

Donde $\partial I_C/\partial V_{CE}$ representa la pendiente de la recta ajustada. A mayor V_A, menor es la variación de I_C con V_{CE}, lo cual indica un comportamiento más ideal.

A continuación se muestran las rectas ajustadas sobre las curvas de salida. Se utilizó un rango de V_{CE} donde el comportamiento fue claramente lineal (por ejemplo, entre 1 y 2,5), evitando zonas de saturación y de efecto resistivo.

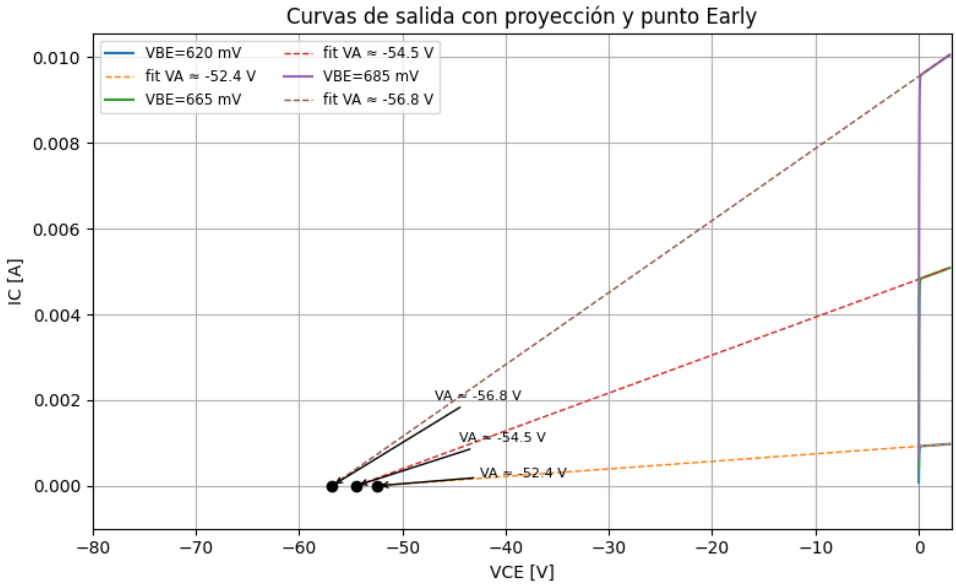


Figura 10: Ajuste lineal de las curvas de salida para estimar V_A y r_o.

Se observa que el ajuste lineal permite proyectar las rectas al eje negativo de tensiones, obteniendo el punto de intersección en $-V_A$, como predice el modelo teórico de Early.

Tabla Comparativa

A partir de los datos de salida se construyó la siguiente tabla con los valores simulados y calculados para cada condición de polarización:

Cuadro 1: Parámetros extraídos de las curvas de salida del TBJ

V_{BE} [V]	I_B [μA]	I_C [mA]	β_0	g_m [S]	V_A [V]	r_o [Ω]
0.620	2.059	0.974	474.34	0.03806	-52.44	56901.74
0.665	11.040	5.084	469.90	0.17920	-54.46	11303.25
0.685	22.567	10.055	463.06	0.32964	-56.82	5948.82

Los resultados obtenidos permiten comparar directamente la ganancia de corriente, la transconductancia y la resistencia de salida en diferentes puntos de operación, estableciendo una base sólida para la modelización del transistor en pequeña señal.

Parte 2 – Modelo de Pequeña Señal

Cálculo de la Ganancia Intrínseca (Modelo Teórico)

La ganancia intrínseca de tensión del transistor en modo activo directo se define como el producto entre la transconductancia g_m y la resistencia de salida r_o:

$$a_v = g_m \cdot r_o \tag{11}$$

Esta expresión surge del modelo de pequeña señal del TBJ en configuración emisor común sin resistencia de carga externa. Se utilizaron los valores de g_m y r_o calculados previamente en la Parte 1 para obtener a_v en los tres puntos de operación correspondientes a los distintos valores de V_{BE}:

Cuadro 2: Resumen de parámetros con ganancia teórica.

V_{BE} [V]	I_B [μA]	I_C [mA]	g_m [S]	r_o [Ω]	a_v (teórico)
0.620	2.059	0.974	0.03806	56901.74	2165.60
0.665	11.040	5.084	0.17920	11303.25	2025.51
0.685	22.567	10.055	0.32964	5948.82	1960.95

Simulación Transitoria

Para validar la ganancia intrínseca obtenida analíticamente, se realizó una simulación transitoria del circuito del tipo amplificador emisor común, utilizando el esquema provisto en la Figura **??**. La fuente de entrada fue una señal senoidal de baja amplitud (1) superpuesta sobre la polarización de base.

Se seleccionaron los valores de V_{BE} e I_C de la tabla [TAB01].

Cálculo de la Ganancia desde la Simulación

Del análisis temporal se extrajeron las señales $v_{be}(t)$ (entrada) y $v_{ce}(t)$ (salida). Como la respuesta del transistor es lineal para señales pequeñas, la ganancia de tensión se puede calcular directamente como el cociente entre las amplitudes pico:

$$a_v^{\text{sim}} = \frac{V_{CE}^{\text{pico}}}{V_{BE}^{\text{pico}}} \tag{12}$$

A continuación se presenta el gráfico con las señales simuladas para los distintos puntos de operación:

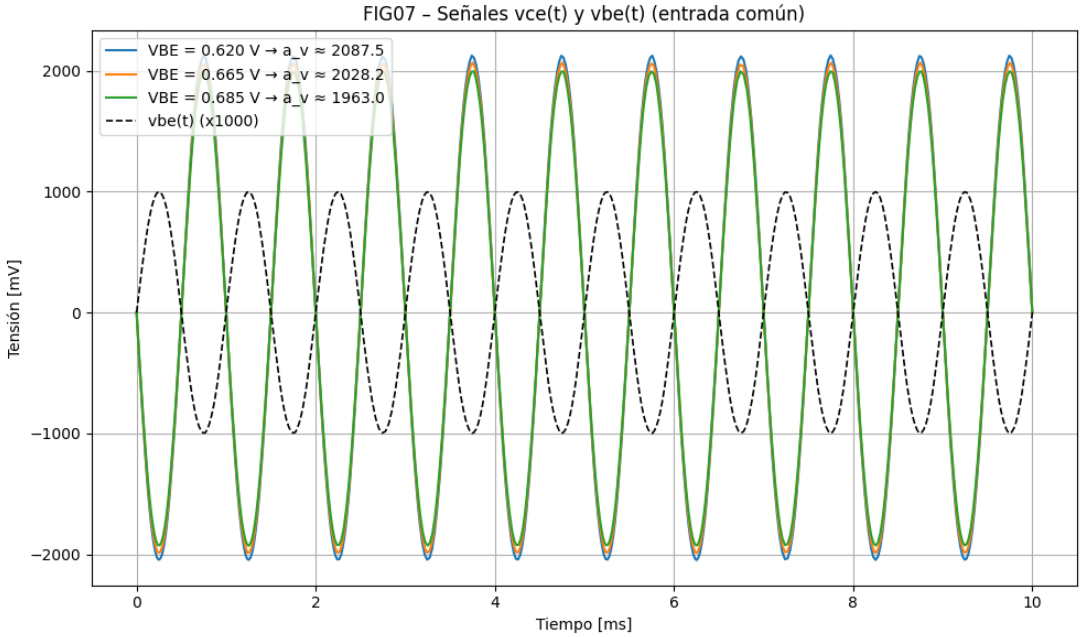


Figura 11: Señales $v_{be}(t)$ y $v_{ce}(t)$ para distintos valores de V_{BE}.

Se escaló la señal $vbe(t)$ para poder apreciar mejor el resultado. Es importante destacar que dicha señal está en contrafase, lo que concuerda con la configuración circuital utilizada.

Cuadro 3: Resumen de parámetros con ganancia teórica y simulada.

V_{BE} [V]	I_B [μA]	I_C [mA]	g_m [S]	r_o [Ω]	a_v (teórico)	a_v (sim)
0.620	2.059	0.974	0.03806	56901.74	2165.60	2087.53
0.665	11.040	5.084	0.17920	11303.25	2025.51	2028.16
0.685	22.567	10.055	0.32964	5948.82	1960.95	1962.99

Los resultados simulados permiten comparar con los obtenidos analíticamente y validar la coherencia del modelo de pequeña señal.

Simulación de Respuesta en Frecuencia

Se simuló el circuito de la Figura **??** en análisis AC para determinar el comportamiento del transistor en frecuencia. La fuente fue una corriente alterna de pequeña señal (1) aplicada a la base, barrida en frecuencia desde 1 hasta 1.

La corriente de colector I_C se midió en cada frecuencia, y se calculó la función de transferencia en corriente:

$$\beta(f) = \left| \frac{I_C(f)}{I_B(f)} \right| \tag{13}$$

Se graficó $\beta(f)$ en escala logarítmica para ambos ejes, observándose una caída característica de filtro paso bajo a alta frecuencia.

Obtención de f_TfT y Tabla Comparativa

La frecuencia de transición f_T se definió como la frecuencia en la que la ganancia de corriente cae a la unidad:

$$\beta(f_T) = 1 \tag{14}$$

Esta frecuencia es un parámetro crítico del TBJ, ya que indica su máxima velocidad de respuesta en aplicaciones de alta frecuencia.

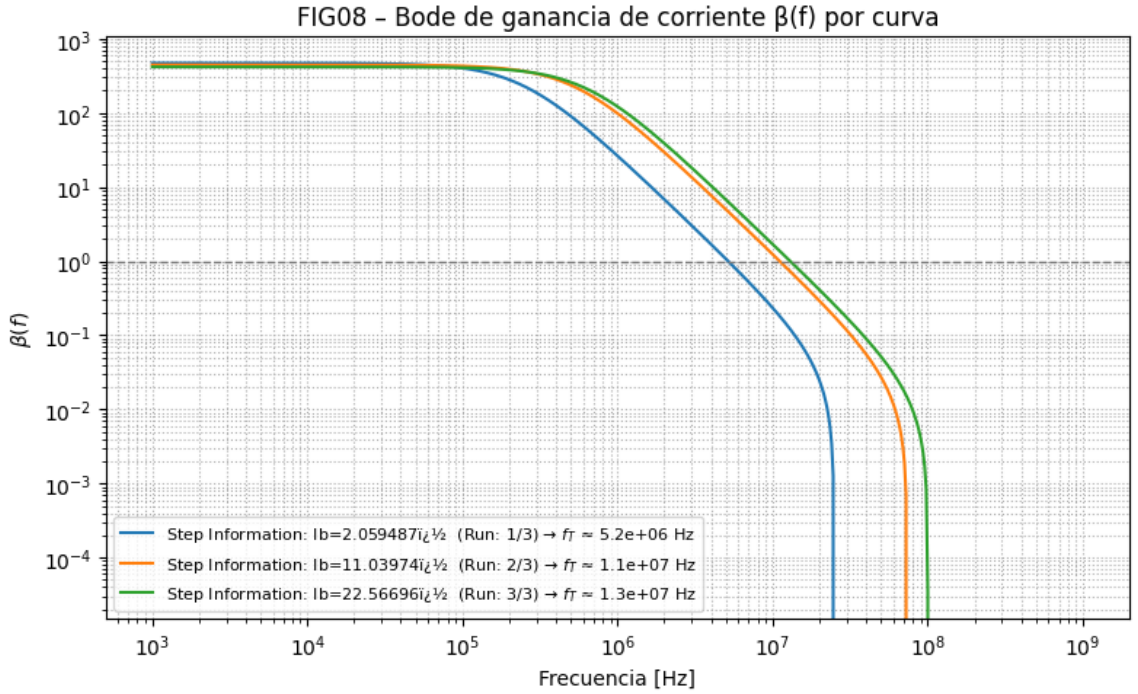


Figura 12: Curvas de Bode de $\beta(f)$ para distintos valores de polarización.

Finalmente, se construyeron las siguientes tablas resumen que comparan los parámetros teóricos y simulados para cada punto de operación:

Cuadro 4: Resumen de parámetros con ganancia teórica y simulada.

V_{BE} [V]	I_B [μA]	I_C [mA]	g_m [S]	r_o [Ω]	a_v (teórico)	a_v (simulado)	β_0	f_T [MHz]
0.620	2.059	0.974	0.03806	56901.74	2165.60	2087.53	470.46	5.163
0.665	11.040	5.084	0.17920	11303.25	2025.51	2028.16	445.34	11.102
0.685	22.567	10.055	0.32964	5948.82	1960.95	1962.99	418.05	12.976

Cuadro 5: Resultados completos del TBJ para distintos valores de V_{BE}.

V_{BE} [V]	I_B [μA]	I_C [mA]	β	g_m [S]	V_A [V]	r_o [Ω]	a_v (teórico)	a_v (simulado)	β_0	f_T [Hz]
0.62	2.06	0.97	474.34	0.04	-52.44	56901.74	2165.60	2087.53	470.46	$5{,}16 \times 10^6$
0.66	11.04	5.08	469.90	0.18	-54.46	11303.25	2025.51	2028.16	445.34	$1{,}11 \times 10^7$
0.68	22.57	10.06	463.06	0.33	-56.82	5948.82	1960.95	1962.99	418.05	$1{,}30 \times 10^7$

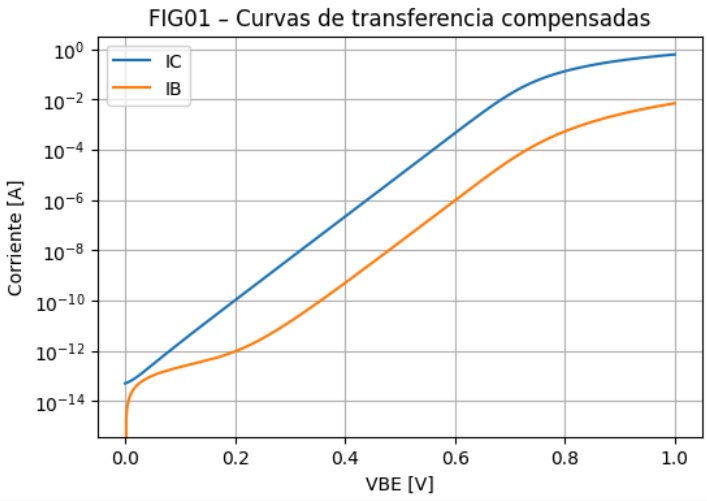
Los valores obtenidos muestran buena coherencia entre el análisis teórico y las simulaciones, confirmando la validez del modelo híbrido-π para pequeñas señales en condiciones normales de polarización.

1. Anexo: Notebook elaborado;

En éste anexo se inlcuye la notebook utilizada para obtener los gráficos y tablas de éste informe:

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

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import linregress
from pathlib import Path

k_B  = 1.380_649e-23
q_e  = 1.602_176_634e-19
T0   = 300.15
V_TH_IDEAL = k_B*T0/q_e  # ≈ 25.85 mV

# --- 1. Cargar CSV con curvas compensadas ---
DATA_PATH = Path("/transfer_curves.txt")   # Cambiá por tu ruta real
df = pd.read_csv(DATA_PATH, sep='\t', comment='*', skip_blank_lines=True)

# Las columnas ya son IB = IB + ICB0 y IC = IC - ICB0
assert {"VBE", "IC", "IB"}.issubset(df.columns)

# --- 2. Graficar curva transferida ---
plt.figure(figsize=(6,4))
plt.semilogy(df["VBE"], df["IC"], label="IC")
plt.semilogy(df["VBE"], df["IB"], label="IB")
plt.xlabel("VBE [V]"); plt.ylabel("Corriente [A]")
plt.title("FIG01 – Curvas de transferencia compensadas")
plt.grid(True, which="both"); plt.legend(); plt.show()


# --- Se guardan datos para la tabla final IC ---
indiceIC1 = np.argmin(np.abs(df["VBE"] - 0.62))
IC1 = df["IC"][indiceIC1]

indiceIC2 = np.argmin(np.abs(df["VBE"] - 0.665))
IC2 = df["IC"][indiceIC2]

indiceIC3 = np.argmin(np.abs(df["VBE"] - 0.685))
IC3 = df["IC"][indiceIC3]

# --- Se guardan datos para la tabla final IB ---
indiceIB1 = np.argmin(np.abs(df["VBE"] - 0.62))
IB1 = df["IB"][indiceIB1]

indiceIB2 = np.argmin(np.abs(df["VBE"] - 0.665))
IB2 = df["IB"][indiceIB2]

indiceIB3 = np.argmin(np.abs(df["VBE"] - 0.685))
IB3 = df["IB"][indiceIB3]
```

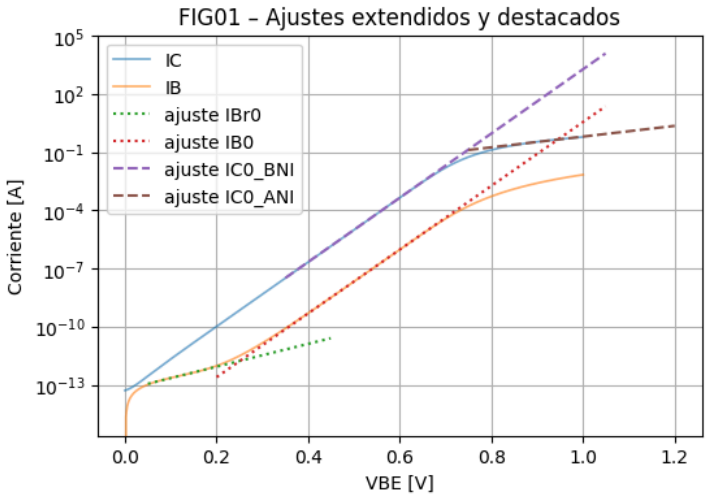



```
# --- 3. Ajustes lineales por regiones ---
def fit_interval(x, y, vmin, vmax, label):
```

```
    msk = (x >= vmin) & (x <= vmax)
    slope, inter, *_ = linregress(x[msk], np.log(y[msk]))
    a, b = slope, inter
    print(f"{label}:  a = {a:.3e}  =>  Vth = {1/a if '1/2' not in label else 1/(2*a):.3e} V")
    return a, b, x[msk], y[msk]


#a) Ibr0 -> IB
#b) Ib0 -> Ib
#c) Ic0(BNI) -> IC
#d) IC0(ANI) -> IC
WINDOWS = {
    "LOW_VBE   (IBr,0)":   (0.05, 0.15),
    "MID_VBE   (IB0)":     (0.40, 0.65),
    "MID_VBE   (IC0_BNI)": (0.40, 0.65),
    "HIGH_VBE  (IC0_ANI)": (0.85, 1.0),
}

fits = {}

# a)
a, b, xfit, yfit = fit_interval(df["VBE"], df["IB"], *WINDOWS["LOW_VBE   (IBr,0)"], "LOW (IBr,0)")
fits["IBr0"] = (np.exp(b), 1/(2*a), a, b, xfit, yfit)

# b)
a, b, xfit, yfit = fit_interval(df["VBE"], df["IB"], *WINDOWS["MID_VBE   (IB0)"], "MID_IB (IB0)")
fits["IB0"] = (np.exp(b), 1/a, a, b, xfit, yfit)

# c)
a, b, xfit, yfit = fit_interval(df["VBE"], df["IC"], *WINDOWS["MID_VBE   (IC0_BNI)"], "MID_IC (IC0_BNI)")
fits["IC0_BNI"] = (np.exp(b), 1/a, a, b, xfit, yfit)

# d)
a, b, xfit, yfit = fit_interval(df["VBE"], df["IC"], *WINDOWS["HIGH_VBE  (IC0_ANI)"], "HIGH (IC0_ANI)")
fits["IC0_ANI"] = (np.exp(b), 1/(2*a), a, b, xfit, yfit)

# Verificación
for k, (_, Vth, *_ ) in fits.items():
    ok = abs(Vth - V_TH_IDEAL)/V_TH_IDEAL <= 0.10
    print(f"{k:10s}  Vth = {Vth*1e3:5.2f} mV  {'OK, Valor dentro de rango' if ok else 'FUERA de ±10 %'}")

print("\n=== PUNTO 3 – Corrientes características estimadas ===\n")

IBr0_val = fits["IBr0"][0]
IB0_val  = fits["IB0"][0]
IC0_BNI_val = fits["IC0_BNI"][0]
IC0_ANI_val = fits["IC0_ANI"][0]

print(f"I_Br,0   = {IBr0_val:.3e} A")
print(f"I_B0     = {IB0_val:.3e} A")
print(f"I_C0     = {IC0_BNI_val:.3e} A   (modo BNI)")
print(f"I_C0'    = {IC0_ANI_val:.3e} A   (modo ANI)")

# --- 4. Dibujar curvas ajustadas con líneas extendidas y contraste ---
plt.figure(figsize=(6,4))

# Curvas originales con más visibilidad
plt.semilogy(df["VBE"], df["IC"], label="IC", alpha=0.6, linewidth=1.2)
plt.semilogy(df["VBE"], df["IB"], label="IB", alpha=0.6, linewidth=1.2)

# Control de cuánto se extienden las líneas de ajuste (por ajuste)
EXTENSIONS = {
    "IBr0":     (0.00, 0.30),   # desde 0.00 V hasta +0.30 V desde el centro
    "IB0":      (0.20, 0.40),
    "IC0_BNI":  (0.05, 0.40),
    "IC0_ANI":  (0.10, 0.30),
}

for name, (_, _, a, b, xfit, _) in fits.items():
    ext_left, ext_right = EXTENSIONS.get(name, (0.05, 0.05))
    v_min = max(0, min(xfit) - ext_left)
    v_max = min(1.2, max(xfit) + ext_right)
    x_ext = np.linspace(v_min, v_max, 300)
    y_ext = np.exp(a * x_ext + b)

    style = "--" if "IC0" in name else ":"
    plt.semilogy(x_ext, y_ext, style, linewidth=1.5, label=f"ajuste {name}")
```

```
plt.xlabel("VBE [V]")
plt.ylabel("Corriente [A]")
plt.title("FIG01 – Ajustes extendidos y destacados")
plt.grid(True, which="both")
plt.legend()
plt.show()
```

```
LOW (IBr,0):  a = 1.356e+01  =>  Vth = 7.377e-02 V
MID_IB (IB0):  a = 3.789e+01  =>  Vth = 2.639e-02 V
MID_IC (IC0_BNI):  a = 3.816e+01  =>  Vth = 2.621e-02 V
HIGH (IC0_ANI):  a = 6.368e+00  =>  Vth = 1.570e-01 V
IBr0        Vth = 36.89 mV  FUERA de ±10 %
IB0         Vth = 26.39 mV  OK, Valor dentro de rango
IC0_BNI     Vth = 26.21 mV  OK, Valor dentro de rango
IC0_ANI     Vth = 78.52 mV  FUERA de ±10 %

=== PUNTO 3 – Corrientes características estimadas ===

I_Br,0   = 5.792e-14 A
I_B0     = 1.283e-16 A
I_C0     = 5.186e-14 A   (modo BNI)
I_C0'    = 1.107e-03 A   (modo ANI)
```

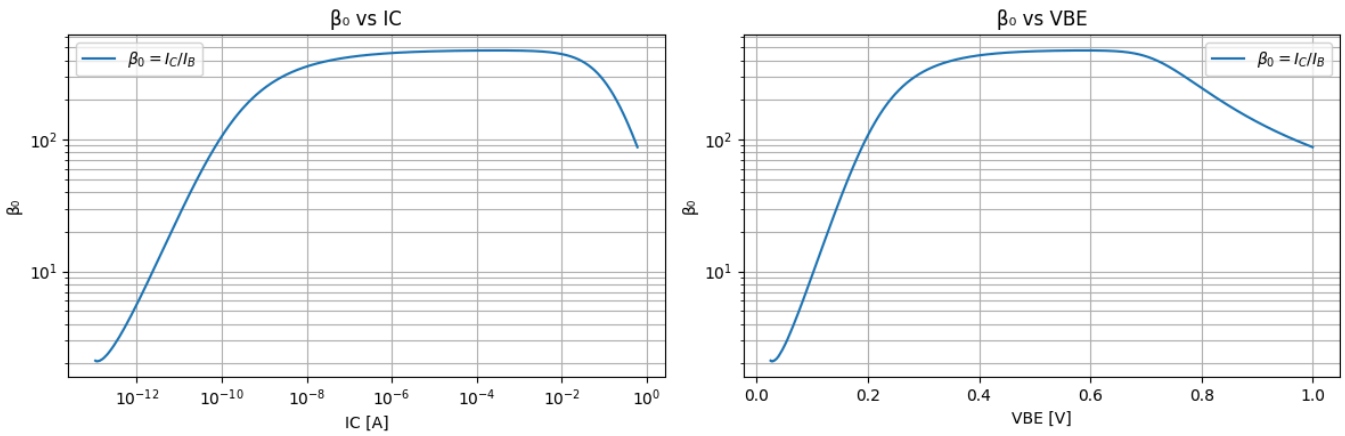



```
# --- 5.  Estimación simplificada del dopaje N_aB usando relación de corrientes ---

ni = 1.45e10  # cm⁻³

IC0_ANI = fits["IC0_ANI"][0]  # modo inverso
IC0_BNI = fits["IC0_BNI"][0]  # modo directo

# Estimación de NaB según la relación mostrada
NaB = (IC0_ANI / IC0_BNI) * (ni / 2)

print("\n=== Estimación simplificada del dopaje en la base (NaB) ===")
print(f"IC0 (ANI) = {IC0_ANI:.3e} A")
print(f"IC0 (BNI) = {IC0_BNI:.3e} A")
print(f"NaB estimado = {NaB:.3e} cm⁻³")

# Evaluación del resultado
if NaB > 1e18:
    print("⚠️ El valor obtenido es lógico pero un poco alto; podría deberse a errores de ajuste o simplificaciones del modelo, principalment
elif NaB < 1e14:
    print("⚠️ El valor obtenido es demasiado bajo. Posiblemente no se cumple la hipótesis del modelo.")
else:
    print("✅ El valor obtenido es razonable y está dentro del rango esperado para la base.")
```

```
=== Estimación simplificada del dopaje en la base (NaB) ===
IC0 (ANI) = 1.107e-03 A
IC0 (BNI) = 5.186e-14 A
NaB estimado = 1.548e+20 cm⁻³
⚠️ El valor obtenido es lógico pero un poco alto; podría deberse a errores de ajuste o simplificaciones del modelo, principalmente a la
```

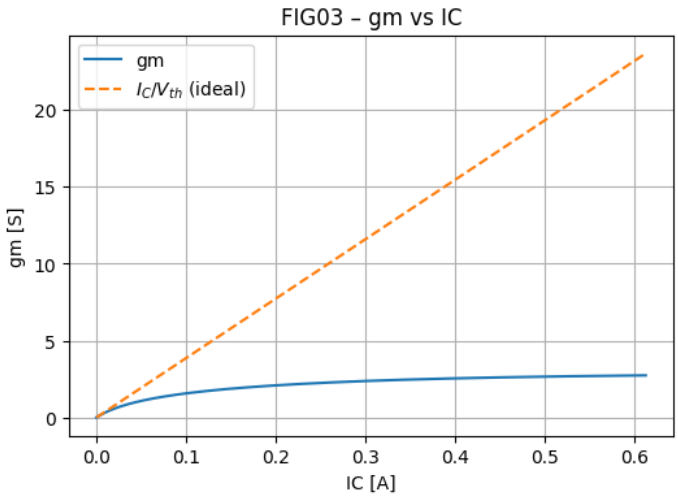
```
# --- PUNTO 6 – β₀ vs IC evitando divergencia cuando IB → 0 ---
```

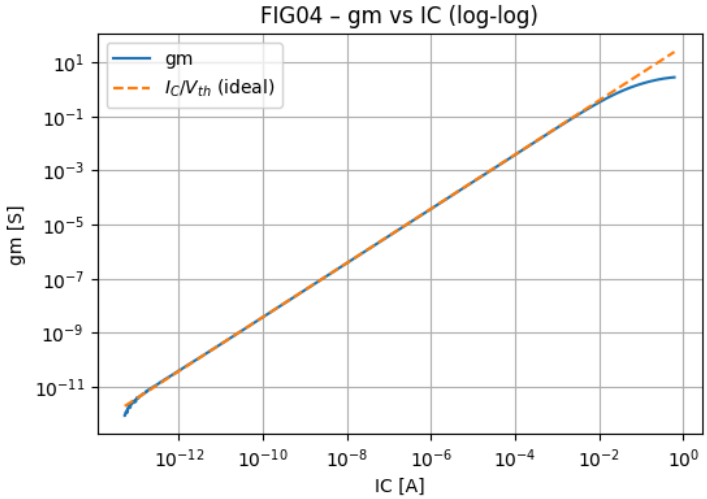
```
# Límite para ajustar posicion del grafico
IB_MIN = 5e-14

# Filtrar valores válidos
mask_valid = df["IB"] > IB_MIN

VBE_valid = df["VBE"][mask_valid]
IC_valid  = df["IC"][mask_valid]
IB_valid  = df["IB"][mask_valid]
beta0_valid = IC_valid / IB_valid

# --- FIG08: Gráficos lado a lado ---
fig, axs = plt.subplots(1, 2, figsize=(12, 4))

# --- β₀ vs IC ---
axs[0].loglog(IC_valid, beta0_valid, label=r"$\beta_0 = I_C / I_B$")
axs[0].set_xlabel("IC [A]")
axs[0].set_ylabel("β₀")
axs[0].set_title("β₀ vs IC")
axs[0].grid(True, which="both")
axs[0].legend()

# --- β₀ vs VBE ---
axs[1].semilogy(VBE_valid, beta0_valid, label=r"$\beta_0 = I_C / I_B$")
axs[1].set_xlabel("VBE [V]")
axs[1].set_ylabel("β₀")
axs[1].set_title("β₀ vs VBE")
axs[1].grid(True, which="both")
axs[1].legend()

plt.tight_layout()
plt.show()

# --- Se guardan datos para la tabla final beta ---
IndiceBeta1 = np.argmin(np.abs(IC_valid - 0.001))
Beta1 = beta0_valid[IndiceBeta1]

indiceBeta2 = np.argmin(np.abs(IC_valid - 0.005))
Beta2 = beta0_valid[indiceBeta2]

indiceBeta3 = np.argmin(np.abs(IC_valid - 0.01))
Beta3 = beta0_valid[indiceBeta3]
```

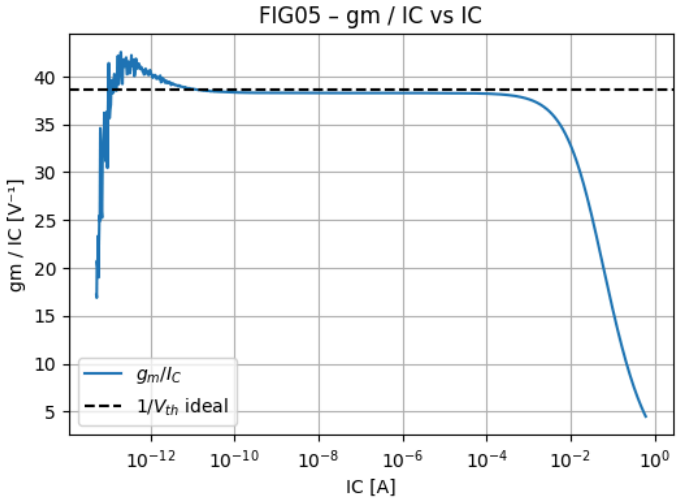



```
# === PUNTO 7 – Cálculo de gm y gráficos FIG03, FIG04, FIG05 ===

# Constante térmica
Vth = 25.85e-3  # V

# Calcular gm ≈ ΔIC / ΔVBE
gm = np.gradient(df["IC"], df["VBE"])  # S
gm_over_IC = gm / df["IC"]

# --- FIG03 – gm vs IC con línea ideal ---
```

```
plt.figure(figsize=(6,4))
plt.plot(df["IC"], gm, label="gm")
plt.plot(df["IC"], df["IC"]/Vth, "--", label=r"$I_C / V_{th}$ (ideal)")
plt.xlabel("IC [A]"); plt.ylabel("gm [S]")
plt.title("FIG03 – gm vs IC")
plt.grid(True); plt.legend(); plt.show()

# --- FIG04 – gm vs IC (log-log) con línea ideal ---
plt.figure(figsize=(6,4))
plt.loglog(df["IC"], gm, label="gm")
plt.loglog(df["IC"], df["IC"]/Vth, "--", label=r"$I_C / V_{th}$ (ideal)")
plt.xlabel("IC [A]"); plt.ylabel("gm [S]")
plt.title("FIG04 – gm vs IC (log-log)")
plt.grid(True, which="both"); plt.legend(); plt.show()

# --- FIG05 – gm/IC vs IC (semilog x) ---
plt.figure(figsize=(6,4))
plt.semilogx(df["IC"], gm_over_IC, label=r"$g_m / I_C$")
plt.axhline(1/Vth, color="k", linestyle="--", label=r"$1 / V_{th}$ ideal")
plt.xlabel("IC [A]"); plt.ylabel("gm / IC [V⁻¹]")
plt.title("FIG05 – gm / IC vs IC")
plt.grid(True, which="both"); plt.legend(); plt.show()

# --- Análisis numérico del valor medio en la meseta ---
IC_min, IC_max = 1e-6, 1e-4
mask = (df["IC"] >= IC_min) & (df["IC"] <= IC_max)
gmIC_mean = gm_over_IC[mask].mean()
gmIC_std  = gm_over_IC[mask].std()

print("\n=== Análisis FIG05 ===")
print(f"Rango evaluado: {IC_min:.0e} A – {IC_max:.0e} A")
print(f"Valor medio de gm/IC ≈ {gmIC_mean:.2f} V⁻¹")
print(f"Valor ideal         ≈ {1/Vth:.2f} V⁻¹")
print(f"Desvío estándar     ≈ {gmIC_std:.2f} V⁻¹")


# --- Se guardan datos para la tabla final Gm ---
indiceGm1 = np.argmin(np.abs(df["IC"] - 0.001))
Gm1 = gm[indiceGm1]

indiceGm2 = np.argmin(np.abs(df["IC"] - 0.005))
Gm2 = gm[indiceGm2]

indiceGm3 = np.argmin(np.abs(df["IC"] - 0.01))
Gm3 = gm[indiceGm3]
```





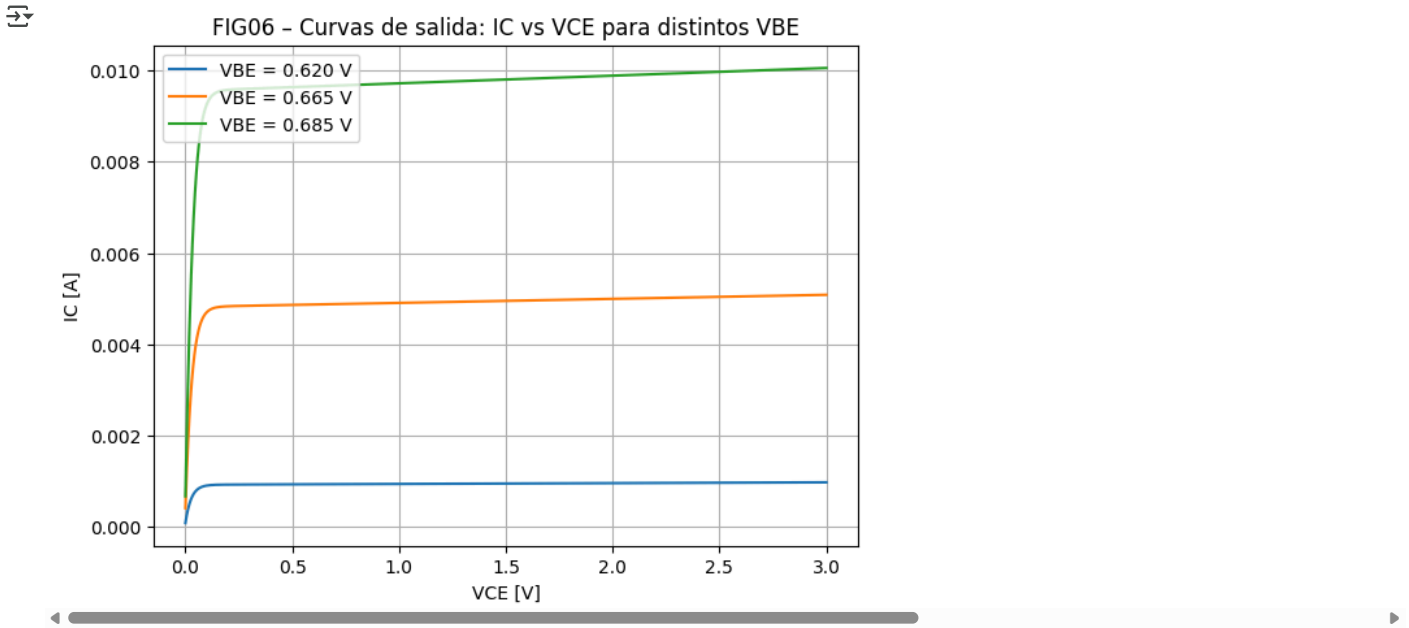



```
=== Análisis FIG05 ===
Rango evaluado: 1e-06 A – 1e-04 A
Valor medio de gm/IC ≈ 38.25 V⁻¹
Valor ideal          ≈ 38.68 V⁻¹
Desvío estándar      ≈ 0.02 V⁻¹
```

```
============================================================
```

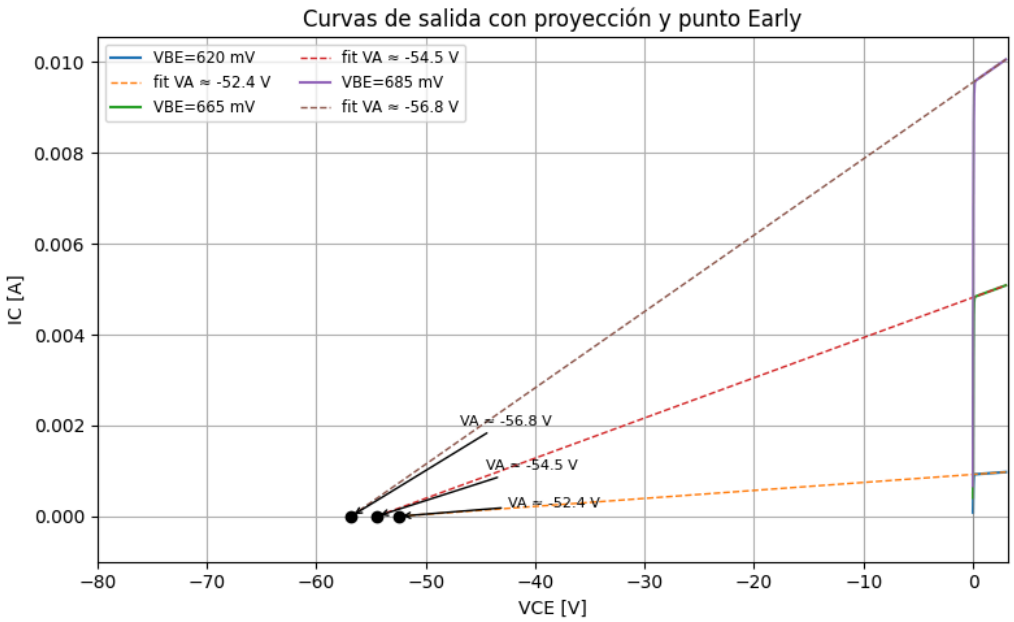
```
import matplotlib.pyplot as plt
import pandas as pd

# === 1. Leer el archivo manualmente para detectar los pasos ===
```

```
runs = []
with open("curvas_salida.txt", "r") as f:
    current_vbe = None
    current_data = []
    for line in f:
        if "Step Information" in line:
            # Almacenar la curva anterior (si existe)
            if current_vbe is not None and current_data:
                runs.append((current_vbe, pd.DataFrame(current_data, columns=["VCE", "IC"])))
                current_data = []
            # Extraer valor de VBE en mV
            try:
                val_str = line.split("Vbe=")[1].split()[0]
                if "m" in val_str:
                    current_vbe = float(val_str.replace("m", "")) / 1000  # pasar a volts
                else:
                    current_vbe = float(val_str)
            except:
                current_vbe = None
        elif line.strip() and not line.startswith("*"):
            # Es una línea de datos
            parts = line.strip().split()
            if len(parts) >= 2:
                try:
                    vce = float(parts[0])
                    ic  = float(parts[1])
                    current_data.append([vce, ic])
                except:
                    continue
    # Guardar la última curva
    if current_vbe is not None and current_data:
        runs.append((current_vbe, pd.DataFrame(current_data, columns=["VCE", "IC"])))

# === 2. Graficar las curvas ===
plt.figure(figsize=(7,5))
for vbe, df_run in runs:
    plt.plot(df_run["VCE"], df_run["IC"], label=f"VBE = {vbe:.3f} V")

plt.xlabel("VCE [V]")
plt.ylabel("IC [A]")
plt.title("FIG07 – Curvas de salida: IC vs VCE para distintos VBE")
plt.grid(True)
plt.legend()
plt.show()
```




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

# === Configuración ===
fit_range = (1.0, 3.0)
```

```
results = []
plt.figure(figsize=(8, 5))

for vbe, df in runs:
    # Filtrar tramo de ajuste
    mask = (df["VCE"] >= fit_range[0]) & (df["VCE"] <= fit_range[1])
    x, y = df.loc[mask, "VCE"].values, df.loc[mask, "IC"].values

    if len(x) < 2:
        continue

    # Ajuste lineal: IC = a * VCE + b
    a, b = np.polyfit(x, y, 1)
    VA = -b / a     # 👈 ¡ESTA es la corrección!
    ro = 1 / a
    results.append(dict(VBE=vbe, a=a, b=b, VA=VA, ro=ro))

    # Curva medida
    plt.plot(df["VCE"], df["IC"], label=f"VBE={vbe*1e3:.0f} mV")

    # Recta proyectada desde VA (negativo) hasta VCE máx
    vce_max = df["VCE"].max()
    vce_fit = np.linspace(VA, vce_max, 200)
    ic_fit = a * vce_fit + b
    plt.plot(vce_fit, ic_fit, '--', linewidth=1, label=f"fit VA ≈ {VA:.1f} V")

    # Punto Early
    plt.plot(VA, 0, 'o', color='black')
    plt.annotate(f"VA ≈ {VA:.1f} V", xy=(VA, 0), xytext=(VA + 10, max(y)*0.2),
                 arrowprops=dict(arrowstyle='->', color='black'),
                 fontsize=8, color='black')

# === Estética ===
plt.axvline(0, color='gray', linewidth=0.5)
plt.xlabel("VCE [V]")
plt.ylabel("IC [A]")
plt.title("Curvas de salida con proyección y punto Early")
plt.grid(True)
plt.xlim(-80, vce_max + 0.2)
plt.ylim(bottom=-0.001)
plt.legend(fontsize="small", ncol=2)
plt.tight_layout()
plt.show()



# === Tabla de resultados ===
tbl = pd.DataFrame(results).set_index("VBE")
print("\nParámetros Early obtenidos:")
print(tbl[["VA", "ro"]].round(2).to_string())

vbes_deseados = [0.62, 0.665, 0.685]
VA_vals = [tbl.loc[vbe, "VA"] for vbe in vbes_deseados]
ro_vals = [tbl.loc[vbe, "ro"] for vbe in vbes_deseados]
data = {
    'Vbe(V)': vbes_deseados,
    'Ib (uA)': [IB1*1e6, IB2*1e6, IB3*1e6],
    'Ic (mA)': [IC1*1e3, IC2*1e3, IC3*1e3],
    'Beta': [Beta1, Beta2, Beta3],
    'gm': [Gm1, Gm2, Gm3],
    'VA': VA_vals,
    'ro': ro_vals,
}
tabla = pd.DataFrame(data)
print('TAB01')
print(tabla.round(6).to_string(index=False))
```



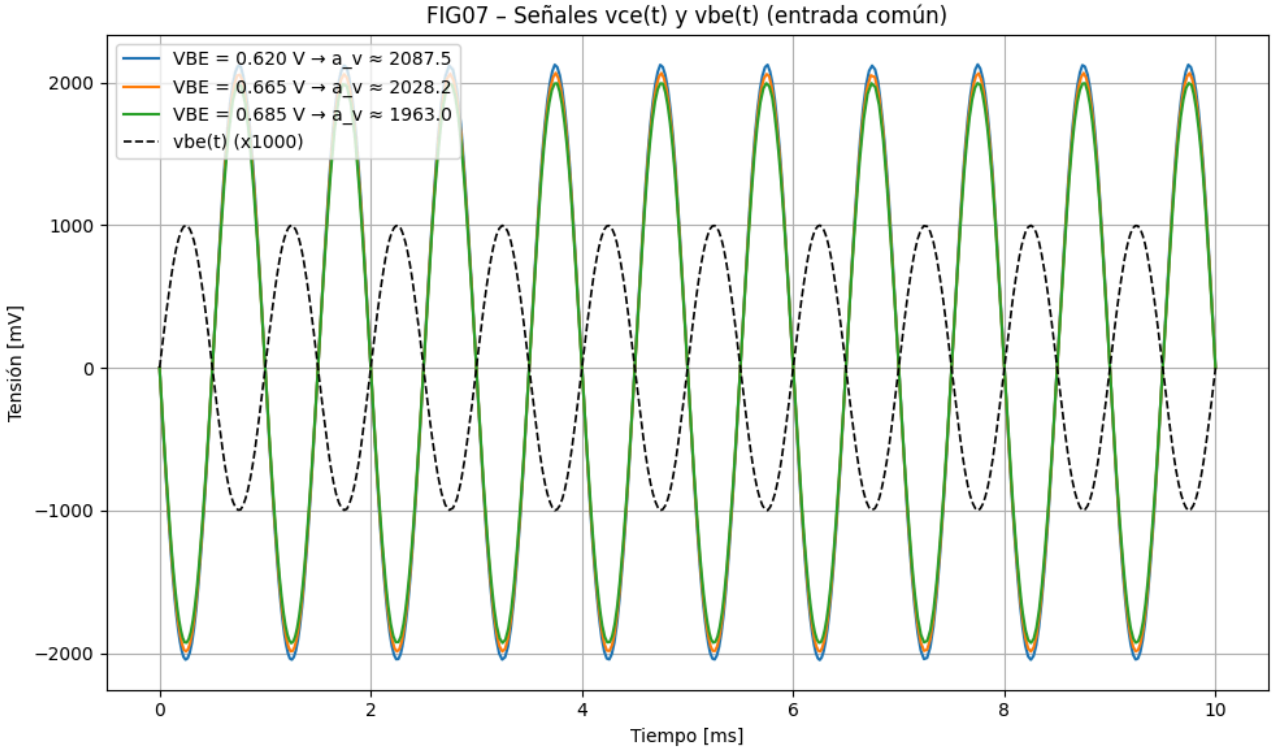

```
Parámetros Early obtenidos:
          VA        ro
VBE
0.620 -52.44  56901.74
0.665 -54.46  11303.25
0.685 -56.82   5948.82
TAB01
 Vbe(V)   Ib (uA)   Ic (mA)       Beta       gm         VA           ro
  0.620  2.059487  0.974310 474.336276 0.038059 -52.440041 56901.744246
  0.665 11.039740  5.083881 469.899203 0.179197 -54.464928 11303.249823
  0.685 22.566960 10.055250 463.064476 0.329636 -56.817877  5948.822423
```

```
# Agregar la ganancia intrínseca a_v
tabla["av"] = tabla["gm"] * tabla["ro"]

print("TAB01 con ganancia intrínseca:")
print(tabla.round(2).to_string(index=False))
```

```
TAB01 con ganancia intrínseca:
 Vbe(V)  Ib (uA)  Ic (mA)   Beta   gm     VA       ro      av
   0.62     2.06     0.97 474.34 0.04 -52.44 56901.74 2165.60
   0.66    11.04     5.08 469.90 0.18 -54.46 11303.25 2025.51
   0.68    22.57    10.06 463.06 0.33 -56.82  5948.82 1960.95
```

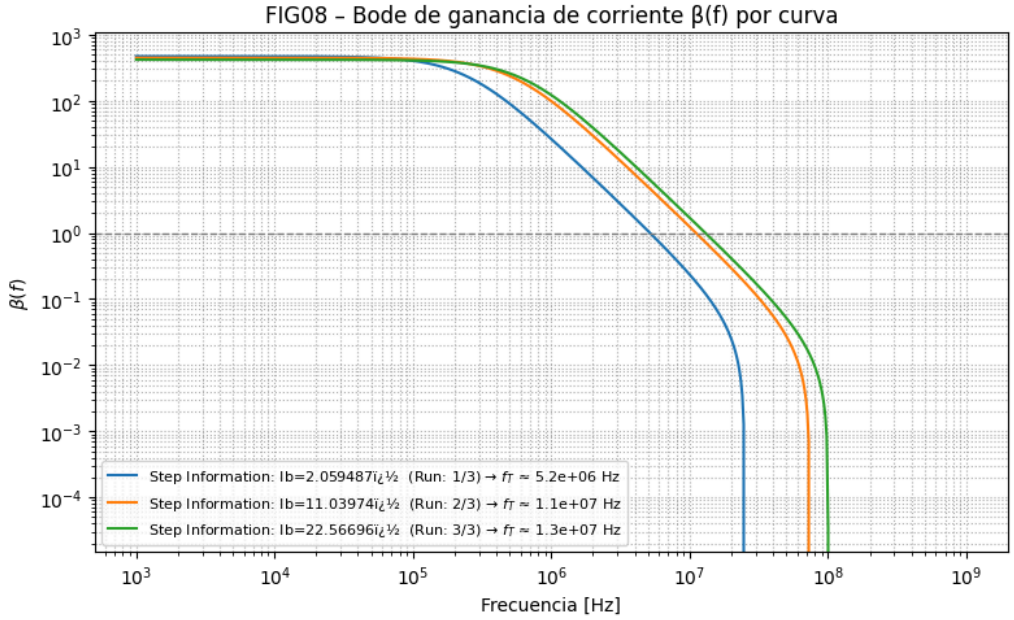
```
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
import numpy as np

# === Archivos con las simulaciones ===
archivos = {
    "VBE = 0.620 V": "62.txt",
    "VBE = 0.665 V": "665.txt",
    "VBE = 0.685 V": "685.txt",
}

# === Parámetros conocidos ===
vbe_amp = 1e-3  # 1 mV de entrada
ganancias = []
plt.figure(figsize=(10, 6))

vbe_comun_t = None
vbe_comun_valores = None

for i, (etiqueta, archivo) in enumerate(archivos.items()):
    path = Path(archivo)
    df = pd.read_csv(path, sep="\t", comment="*", skip_blank_lines=True)

    # Detectar columnas
    cols = df.columns.str.lower()
```

```
    tiempo_col = [col for col in cols if "time" in col][0]
    vbe_col = [col for col in cols if "vbe" in col][0]
    vce_col = [col for col in cols if "vce" in col][0]

    t = df[tiempo_col].values
    vbe = df[vbe_col].values
    vce = df[vce_col].values

    # Guardar vbe(t) solo una vez
    if vbe_comun_t is None:
        vbe_comun_t = t
        vbe_comun_valores = vbe

    # Calcular amplitud de vce (peak-to-peak / 2)
    vce_amp = (np.max(vce) - np.min(vce)) / 2
    av = vce_amp / vbe_amp
    ganancias.append((etiqueta, av))

    # Graficar vce(t)
    plt.plot(t * 1e3, vce * 1e3, label=f"{etiqueta} → a_v ≈ {av:.1f}")

# Superponer vbe(t)
plt.plot(vbe_comun_t * 1e3, vbe_comun_valores * 1e3, '--k', linewidth=1.2, label="vbe(t) (x1000)")

# Estética
plt.xlabel("Tiempo [ms]")
plt.ylabel("Tensión [mV]")
plt.title("FIG07 – Señales vce(t) y vbe(t) (entrada común)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Mostrar tabla
tabla_av = pd.DataFrame(ganancias, columns=["VBE", "a_v"])
print("\nGanancia intrínseca (a_v) calculada desde simulación transitoria:")
print(tabla_av.to_string(index=False))
```




```
Ganancia intrínseca (a_v) calculada desde simulación transitoria:
          VBE       a_v
VBE = 0.620 V 2087.5295
VBE = 0.665 V 2028.1640
VBE = 0.685 V 1962.9940
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from io import StringIO

# Leer todo el archivo
with open("Bode.txt", encoding="latin1") as f:
    lines = f.readlines()

# Separar por curvas usando Step Information
curvas = []
etiquetas = []
current_data = []
for line in lines:
    if "Step Information" in line:
        if current_data:
            curvas.append(current_data)
            current_data = []
        etiquetas.append(line.strip())
    elif line.strip() and line[0].isdigit():
        current_data.append(line)
if current_data:
    curvas.append(current_data)

# Procesar cada curva
ft_resultados = []
plt.figure(figsize=(8, 5))

for idx, (datos, nombre) in enumerate(zip(curvas, etiquetas)):
    df = pd.read_csv(StringIO("".join(datos)), sep="\t", names=["Freq", "Ic_complex"])
    df["Freq"] = df["Freq"].astype(float)
    df["beta"] = df["Ic_complex"].str.extract(r'([+-]?\d+\.\d+e?[+-]?\d*)')[0].astype(float)

    # Limitar a 1 GHz
    df = df[df["Freq"] <= 1e9]

    # Eliminar discontinuidades en baja frecuencia
    ref_beta = df["beta"].iloc[:10].median()
    tolerancia = 0.05 * ref_beta
    filtro_baja = df["Freq"] < 5e4
    df = df[~(filtro_baja & (np.abs(df["beta"] - ref_beta) > tolerancia))]

    # Extraer arrays
    f = df["Freq"].values
    beta = df["beta"].values
    if len(f) < 2:
        continue

    # Interpolar para obtener f_T (frecuencia tal que beta = 1)
    try:
        interp = interp1d(beta[::-1], f[::-1], bounds_error=False, fill_value="extrapolate")
        f_T = float(interp(1.0))
    except:
        f_T = np.nan

    # Guardar resultados
    ft_resultados.append((nombre, ref_beta, f_T))

    # Graficar
    plt.loglog(f, beta, label=fr"{nombre} → $f_T$ ≈ {f_T:.1e} Hz")

# Estética
plt.axhline(1, color="gray", linestyle="--", linewidth=1)
plt.xlabel("Frecuencia [Hz]")
plt.ylabel(r"$\beta(f)$")
plt.title("FIG08 – Bode de ganancia de corriente β(f) por curva")
plt.grid(True, which="both", linestyle=":")
plt.legend(fontsize=8)
plt.tight_layout()
plt.show()

# Mostrar tabla
tabla_ft = pd.DataFrame(ft_resultados, columns=["Curva", "β₀", "f_T [Hz]"])
print("TABLA – f_T para cada curva:")
print(tabla_ft.to_string(index=False))
```




```
TABLA – f_T para cada curva:
                                    Curva         β₀     f_T [Hz]
Step Information: Ib=2.059487ï¿½  (Run: 1/3) 470.460026 5.162716e+06
Step Information: Ib=11.03974ï¿½  (Run: 2/3) 445.339632 1.110202e+07
Step Information: Ib=22.56696ï¿½  (Run: 3/3) 418.051150 1.297589e+07
```

```
# === Tabla de resultados ===
tbl = pd.DataFrame(results).set_index("VBE")

vbes_deseados = [0.62, 0.665, 0.685]
VA_vals = [tbl.loc[vbe, "VA"] for vbe in vbes_deseados]
ro_vals = [tbl.loc[vbe, "ro"] for vbe in vbes_deseados]
data = {
    'Vbe(V)': vbes_deseados,
    'Ib (uA)': [IB1*1e6, IB2*1e6, IB3*1e6],
    'Ic (mA)': [IC1*1e3, IC2*1e3, IC3*1e3],
    'Beta': [Beta1, Beta2, Beta3],
    'gm': [Gm1, Gm2, Gm3],
    'VA': VA_vals,
    'ro': ro_vals,
}
tabla = pd.DataFrame(data)
print('TAB01')
print(tabla.round(6).to_string(index=False))
```

```
TAB01
 Vbe(V)   Ib (uA)   Ic (mA)       Beta       gm         VA           ro
  0.620  2.059487  0.974310 474.336276 0.038059 -52.440041 56901.744246
  0.665 11.039740  5.083881 469.899203 0.179197 -54.464928 11303.249823
  0.685 22.566960 10.055250 463.064476 0.329636 -56.817877  5948.822423
```

```
import pandas as pd

# === Datos base ===
vbes_deseados = [0.62, 0.665, 0.685]

# Valores provenientes del análisis anterior (ya definidos previamente)
IBs = [IB1 * 1e6, IB2 * 1e6, IB3 * 1e6]         # µA
ICs = [IC1 * 1e3, IC2 * 1e3, IC3 * 1e3]         # mA
betas = [Beta1, Beta2, Beta3]
gms = [Gm1, Gm2, Gm3]

# === VA y ro desde tabla 'results'
tbl = pd.DataFrame(results).set_index("VBE")
VA_vals = [tbl.loc[vbe, "VA"] for vbe in vbes_deseados]
ro_vals = [tbl.loc[vbe, "ro"] for vbe in vbes_deseados]

# === armar tabla base
tabla = pd.DataFrame({
    'VBE [V]': vbes_deseados,
    'IB [µA]': IBs,
```

```
    'IC [mA]': ICs,
    'β': betas,
    'gm': gms,
    'VA [V]': VA_vals,
    'ro [Ω]': ro_vals,
})

# === a_v teórico
tabla["a_v teórico"] = tabla["gm"] * tabla["ro [Ω]"]

# === a_v simulado desde tabla_av
tabla_av = pd.DataFrame(ganancias, columns=["VBE", "a_v"])
tabla_av["VBE"] = tabla_av["VBE"].astype(str).str.extract(r'([\d.]+)').astype(float)
```