trabajo_pompa

June 9, 2024

1 Ejercicio de Optimización: Pompa de jabón

Trabajo realizado por el **Grupo 1**

- Landolfi Cano, Silvia
- López Gallo, Ismael
- López García, Álvaro
- Rodríguez de Frutos, Pablo

1.1 Introducción

En este documento se presenta el problema de optimización sobre el caso de una pompa de jabón apoyada entre dos circunferencias paralelas y coaxiales, como se puede ver en la siguiente imagen:

Figura 1. Pompa de jabón apoyada entre dos circunferencias paralelas y coaxiales

En la forma de equilibrio que adopta la superficie libre de este fluido es aquella que minimiza la energía total del sistema para el caso con las restricciones que se tengan. En esta energía, se han de tomar en consideración, energías como pueden ser:

- La energía debida a la **tensión superficial**, proporcional al área.
- La **energía potencial**, debida a la gravedad y/o a la rotación como sólido rígido.

Este problema presenta una solución analítica para uno de los problemas más sencillos: la **catenoide**.

Suponiendo una superficie axilsimétrica, con un radio dado por $\rho = F(z)$, la energía superficial es proporcional (siendo la constante de proporcionalidad la energía por unidad de área) a la superficie, siendo esta proporcional (una vez adimensionalizada adecuadamente) a:

$A = \int_0^1 F\sqrt{1 + \left(\frac{dF}{dz}\right)^2} dz$

Donde se ha escogido la adimensionalización para que la distancia entre las superficies sea la unidad.

1.2 Planteamiento del problema

En primer lugar, se van a importar las bibliotecas necesarias para el cálculo del problema. Algunas de las más relevantes son:

- `numpy`: principal librería de Python que permite trabajar con arrays e incluye funciones de alto nivel para operar con ellos.

- scipy: que proporciona los algoritmos de optimización empleados para el problema de estudio.
- matplotlib: empleada para representar gráficamente los resultados.
- time: para calcular el tiempo empleado en correr los fragmentos de código.
- pygad: soporte para implementar algorimos genéticos de optimización.
- pyswarms: desarrollada por el MIT para implementar el algoritmo de optimización *Particle Swarm*.
- simanneal: permite la implementación del algoritmo de optimización *Simulated Annealing*.

[1]:
```
import numpy as np
import autograd.numpy as anp
from scipy.optimize import minimize, fsolve, differential_evolution,␣
↪NonlinearConstraint
from scipy.integrate import solve_bvp
import matplotlib.pyplot as plt
import time
import pygad
import pyswarms as ps
from simanneal import Annealer
import seaborn as sns
```

```
2024-06-09 17:46:11,355 - numexpr.utils - INFO - NumExpr defaulting to 8
threads.
```

[2]:
```
# Configuración global de Matplotlib
plt.rcParams.update({
    'text.usetex': True,  # Usar LaTeX para el texto
    'font.family': 'serif',  # Fuente serif
    # 'figure.figsize': (10, 6),  # Tamaño de la figura
    'axes.labelsize': 12,  # Tamaño de las etiquetas de los ejes
    'axes.titlesize': 14,  # Tamaño del título
    'legend.fontsize': 12,  # Tamaño de la leyenda
    'xtick.labelsize': 10,  # Tamaño de las etiquetas del eje x
    'ytick.labelsize': 10,  # Tamaño de las etiquetas del eje y
    'axes.grid': True,  # Habilitar la cuadrícula
    'grid.alpha': 0.75,  # Transparencia de la cuadrícula
    'grid.linestyle': '--'  # Estilo de la línea de la cuadrícula
})

# Configuración de Seaborn
sns.set_context('paper')
sns.set_style('whitegrid')
```

1.2.1 Problema numérico de optimización

A continuación, se define el funcional que se busca minimizar. En el caso del problema que se está considerando, se trata de la superficie del fluido entre ambos soportes, cuya expresión se desarrolla en la introducción.

[3]:
```
def area_func(F):
    Famp = anp.append(anp.array([F0]), anp.append(F, F1))
    n = anp.size(Famp)
    delta_z = 1 / (n - 1)

    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z # esquema adelantado para el␣
 ↪cálculo de la derivada
        # calcular el valor de la integral
        integrando = Famp[i] * anp.sqrt(1 + F_der**2)
        integral += integrando * delta_z

    return integral
```

Se plantea la condición de contorno para el caso en el que ambas circunferencias sean de igual radio, que resulta ser:

$F(z = 0) = F(z = 1) = F_0$

Para el primer estudio, se escoge arbitrariamente $F_0 = 1$. Más adelante, se estudiará cómo afecta la variación de este parámetro al cálculo de la solución.

[4]:
```
F_0 = 1
F0 = F_0
F1 = F_0
```

En una primera aproximación, se estudiará para una partición equiespaciada y con un número n arbitrario de puntos.

El valor `F_init` (vector de valores que arranca la iteración) se inicializa para el caso en el que \$ F(z) = F_0 , $\forall z \in [0, 1]$\$, $es decir, en la posicin en la que la superficie formara un cilindro recto entre ambos soportes$.

Además del sentido físico que pueda tener considerar cómo evoluciona la pompa desde una posición próxima a la de equilibrio, interesa imponer estos valores de arranque por la proximidad de la solución a estos valores del radio, especialmente en las proximidades de los soportes.

Posteriormente se comentará el efecto de estas condiciones de arranque a la solución del problema.

[5]:
```
n = 20
F_init = np.empty(n-2)
F_init.fill(F_0)
```

1.2.2 Solución analítica del problema físico

Volviendo a problema físico, este tiene una solución analítica que, en el caso 2D, toma la forma de una catenaria. Esta es aplicable al problema sin restricciones de volumen, por lo que se va a emplear para comparar las soluciones numéricas obtenidas.

Así, esta curva sigue la siguiente ecuación:

$F(z) = a\cosh(\frac{z+k}{a})$

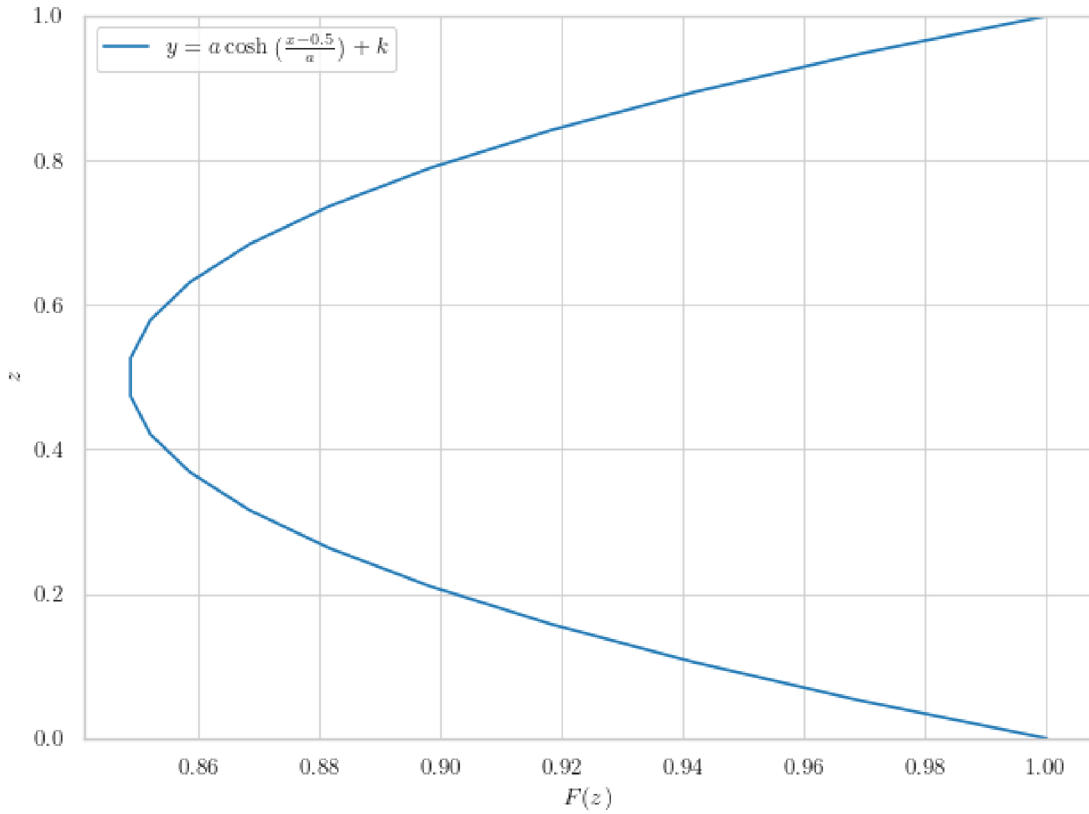
Donde h, a y k son parámetros que dependen del problema que se esté considerando, por lo que habrá que resolverlos cada vezque se modifiquen las condiciones del problema. Solo por considerar el intervalo $z \in [0, 1]$, se fija el parámetro $h = 0.5$, al ser este el punto medio del intervalo.

A continuación se plantea como ejemplo el caso más sencillo, que es el que se empleará en la mayoría de análisis a lo largo del trabajo.

[6]:
```python
eps = 0 # soportes iguales
def params_catenaria(params, eps=eps):
    a, k = params
    eq1 = F0 - a * np.cosh(k / a)
    eq2 = F1 - a * np.cosh( (1 + k) / a)
    return [eq1, eq2]

initial_guess = [1.0, 0.0]

a_sol, k_sol = fsolve(params_catenaria, initial_guess)

z = np.linspace(0, 1, n)

def catenaria(z):
    return a_sol * np.cosh((z + k_sol) / a_sol)

F_cat = catenaria(z)
a_sol, k_sol
```

[6]:
```
(0.8483379380949795, -0.4999999999999956)
```

[7]:
```python
plt.figure()
plt.plot(F_cat, z, label=r'$y = a \cosh\left(\frac{x - 0.5}{a}\right) + k$')
# plt.title('Catenaria en el caso de $F(0) = F(1) = F_0 = 1$')
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.legend()
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
plt.savefig('Figuras/catenaria.pdf', format='pdf')
plt.show()
```

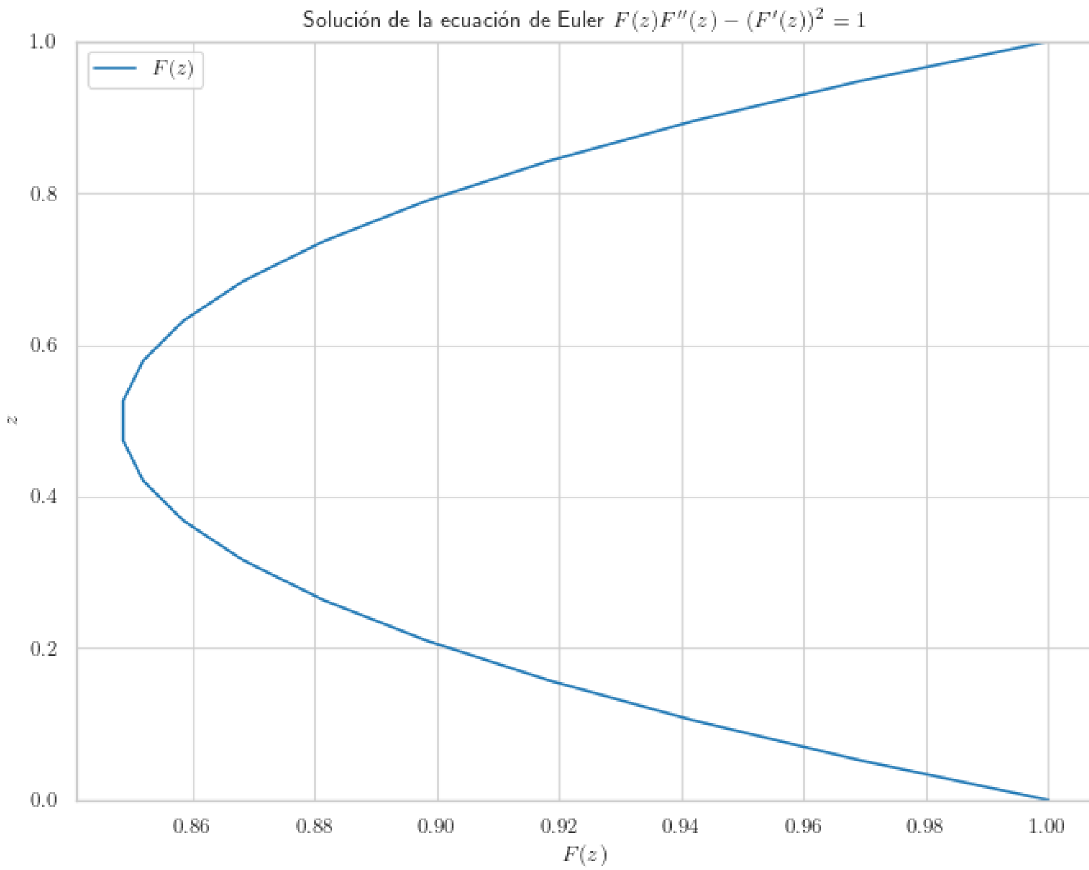


1.2.3 Solución de la ecuación de Euler

[8]:
```
# Definir el sistema de ecuaciones diferenciales
def euler_system(z, y):
    y0, y1 = y
    dydz = np.vstack((y1, (y1**2 + 1) / y0))
    return dydz

# Definir las condiciones de frontera
def boundary_conditions(ya, yb):
    return np.array([ya[0] - F0, yb[0] - F0])  # F(0) = 1 y F(1) = 2, por ejemplo

# Intervalo de integración
z = np.linspace(0, 1, n)  # Ajusta según sea necesario

# Suposición inicial para F y F' (se requiere para solve_bvp)
y_initial_guess = np.zeros((2, z.size))
y_initial_guess[0] = 1 + z  # Suposición inicial simple

# Resolver la EDO con condiciones de frontera
sol_euler = solve_bvp(euler_system, boundary_conditions, z, y_initial_guess)
```

```
# Verificar si la solución se ha encontrado correctamente
if sol_euler.status != 0:
    print("Advertencia: La solución puede no ser precisa.")

# Extraer las soluciones
z = sol_euler.x
F_euler = sol_euler.y[0]
```

[9]:
```
# Graficar la solución
plt.figure(figsize=(8, 6))
plt.plot(F_euler, z, label='$F(z)$')
plt.ylabel('$z$')
plt.xlabel('$F(z)$')
plt.title('Solución de la ecuación de Euler $F(z) F\'\'(z) - (F\'(z))^2 = 1$')
plt.ylim(0, 1)
plt.legend()
plt.show()
```



[10]:
```
error_euler = np.linalg.norm(F_euler - F_cat) / np.linalg.norm(F_cat)
error_euler
```

[10]: 2.4178363133322142e-06

1.3 Problema sin restricciones, con soportes iguales.

Pese a que inicialmente se considere un problema sin restricciones adicionales, no se puede ignorar el sentido físico del funcional que se quiere optimizar: el radio o distancia del eje a la superficie libre del fluído, magnitud que, por definición, es siempre positiva. Así, se impone esta frontera en el primer cálculo de la solución del problema.

[113]:
```
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

1.3.1 Minimización con un método basado en gradiente

En primer lugar, dentro de los método de tipo gradiente, se considerará el método de *Sequential Least Squares Programming*. La explicación del método de desarrollará más adelante en el apartado dedicado a ver la influencia de los métodos de optimización en el resultado del problema.

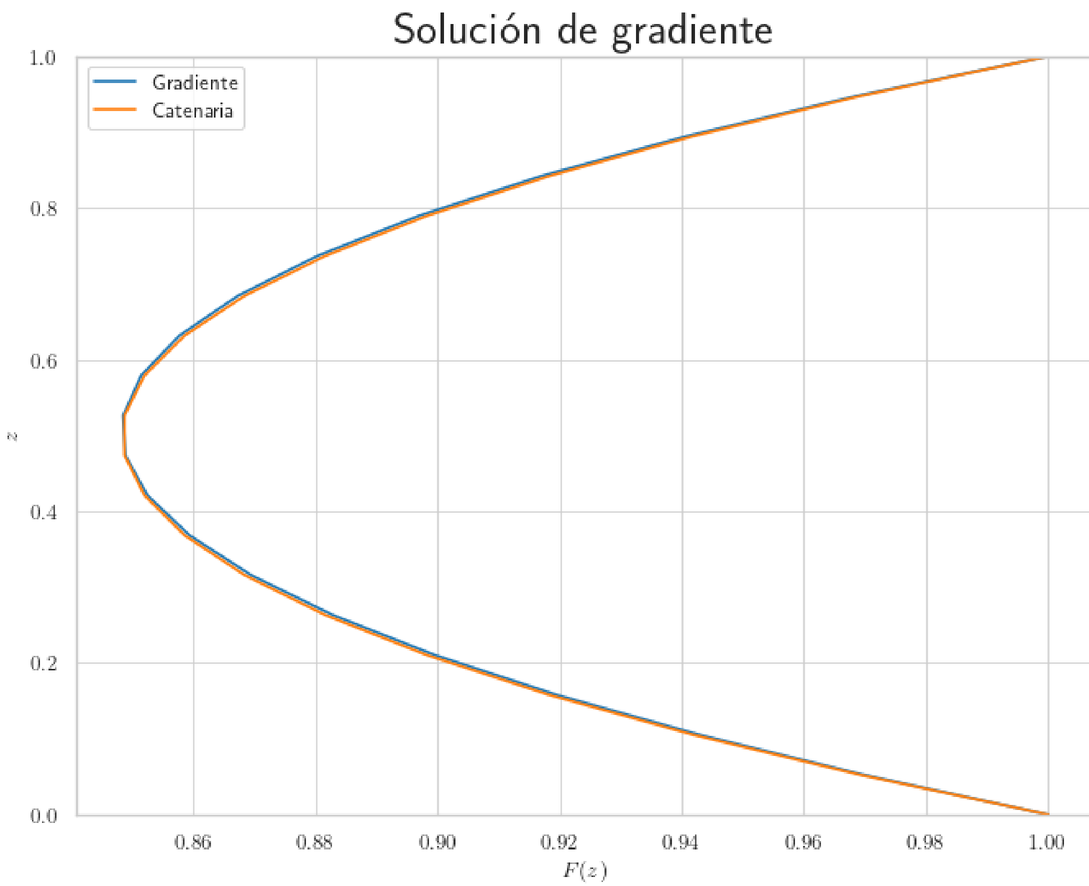
Para más información sobre las librerías empleadas para el cálculo con algoritmos de optimización de tipo gradiente, esta se puede consultar en este enlace.

[114]:
```
time_start = time.time()
sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()

time_grad = time_end - time_start
sol_grad
```

[114]:
```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.9537487850868308
       x: [ 9.697e-01  9.429e-01 ...  9.408e-01  9.684e-01]
     nit: 21
     jac: [-3.795e-04  4.508e-04 ... -3.908e-04  1.709e-04]
    nfev: 429
    njev: 21
```

[115]:
```
Famp = np.concatenate(([F0], sol_grad.x, [F1]))

F_cat = catenaria(z)

# Crear la gráfica
plt.figure(figsize=(8, 6))
plt.plot(Famp, z, label='Gradiente')
```

```
plt.plot(F_cat, z, label='Catenaria')
plt.ylabel('$z$')
plt.xlabel('$F(z)$')
plt.legend()
plt.ylim(0, 1)
# plt.xlim(0, 1)
plt.title('Solución de gradiente', fontsize=20)
plt.savefig('Figuras/Presentación/comp_grad_cat.pdf', format='pdf',␣
 ↪bbox_inches='tight', pad_inches=0.1)
# plt.savefig('Figuras/comp_grad_cat.pdf', format='pdf')
plt.grid(True)
plt.show()
```



A partir de la norma cuadrática de la diferencia entre la solución analítica y la solución calculada se va a calcular el error de la solución. Así, se va a tomar a partir de ahora el siguiente valor de error como referencia para los apartados siguientes. Por tanto, se va a considerar que una aproximación con un error del orden del siguiente es aceptable.

[116]:
```python
error = np.linalg.norm(Famp - F_cat)/np.linalg.norm(F_cat)
print('El error cuadrático medio de la aproximación es:', error)
print('Tiempo de ejecución:', time_end - time_start)
```

```
El error cuadrático medio de la aproximación es: 0.0010218191336893907
Tiempo de ejecución: 0.11530470848083496
```

Influencia del radio, F_0. En este apartado se va a considerar la variación del radio de los dos soportes, F_0, bajo la condición de que se mantenga igual en ambos, es decir, manteniendo $F(0) = F(1) = F_0$. Por otro lado, se congelan el resto de parámetros, manteniéndose igual que en el primer estudio.

[142]:
```python
n = 20
F_init = np.empty(n-2)
z = np.linspace(0, 1, n)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

- $F_0 = 1$

[143]:
```python
F_0 = 1
F0 = F_0
F1 = F_0
F_init.fill(F_0)

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f1 = catenaria(z)

sol_gradf1 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Fampf1 = np.concatenate(([F0], sol_gradf1.x, [F1]))

error_f1 = np.linalg.norm(Fampf1 - cat_f1)/np.linalg.norm(cat_f1)
print('El error para F_0=1 es:', error_f1)
```

```
El error para F_0=1 es: 0.0010218191336893907
```

- $F_0 = 2$

[144]:
```python
F_0 = 2
F0 = F_0
F1 = F_0
F_init.fill(F_0)

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f2 = catenaria(z)

sol_gradf2 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Fampf2 = np.concatenate(([F0], sol_gradf2.x, [F1]))
```

```
error_f2 = np.linalg.norm(Fampf2 - cat_f2)/np.linalg.norm(cat_f2)
print('El error para F_0=2 es:', error_f2)
```

```
El error para F_0=2 es: 5.469805073531967e-05
```

- $F_0 = 5$

[145]:
```
F_0 = 5
F0 = F_0
F1 = F_0
F_init.fill(F_0)

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f5 = catenaria(z)

sol_gradf5 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Fampf5 = np.concatenate(([F0], sol_gradf5.x, [F1]))

error_f5 = np.linalg.norm(Fampf5 - cat_f5)/np.linalg.norm(cat_f5)
print('El error para F_0=5 es:', error_f5)
```

```
El error para F_0=5 es: 0.00014188955706424418
```

- $F_0 = 0.8$

[146]:
```
F_0 = 0.8
F0 = F_0
F1 = F_0
F_init.fill(F_0)

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f08 = catenaria(z)

sol_gradf08 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Fampf08 = np.concatenate(([F0], sol_gradf08.x, [F1]))

error_f08 = np.linalg.norm(Fampf08 - cat_f08)/np.linalg.norm(cat_f08)
print('El error para F_0=0.8 es:', error_f08)
```

```
El error para F_0=0.8 es: 0.006038139196062663
```

- $F_0 = 0.7$

[147]:
```
F_0 = 0.7
F0 = F_0
F1 = F_0
F_init.fill(F_0)

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
```

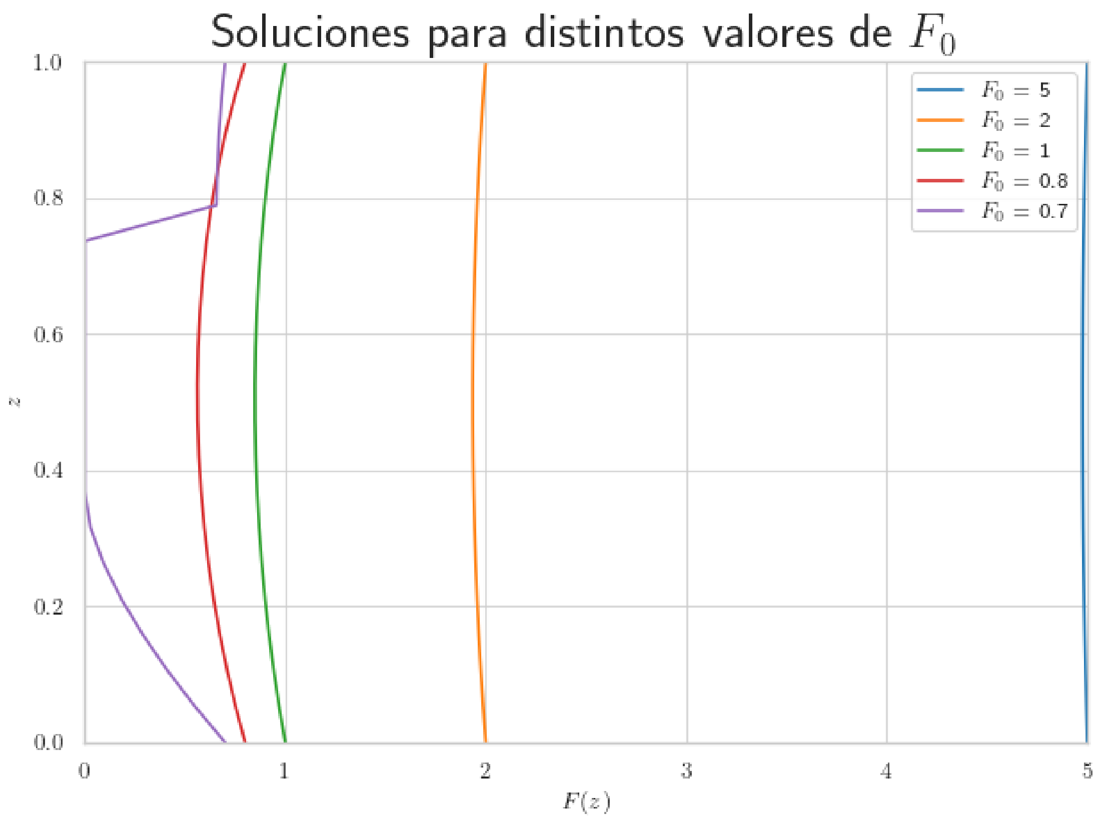
```
cat_f07 = catenaria(z)

sol_gradf07 = minimize(area_func, F_init, bounds=bounds, method='SLSQP') # SLSQP␣
↪no converge -> COBYLA
Fampf07 = np.concatenate(([F0], sol_gradf07.x, [F1]))

error_f07 = np.linalg.norm(Fampf07 - cat_f07)/np.linalg.norm(cat_f07)
print('El error para F_0=0.7 es:', error_f07)
```

```
C:\Users\ismag\AppData\Local\Temp\ipykernel_19584\2021984694.py:6:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  a_sol, k_sol = fsolve(params_catenaria, initial_guess)
```

```
El error para F_0=0.7 es: 0.6003927362443426
```

[148]:

```
plt.figure()
plt.plot(Fampf5, z, label='$F_0$ = 5')
plt.plot(Fampf2, z, label='$F_0$ = 2')
plt.plot(Fampf1, z, label='$F_0$ = 1')
plt.plot(Fampf08, z, label='$F_0$ = 0.8')
plt.plot(Fampf07, z, label='$F_0$ = 0.7')
# plt.plot(Fampf06, z, label='$F_0$ = 0.6')
plt.legend()
# plt.xscale('log')
plt.ylabel('$z$')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
plt.xlim(0, 5)
plt.title('Soluciones para distintos valores de $F_0$', fontsize=20)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.savefig('Figuras/sol_radios.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_radios.pdf', format='pdf')
plt.show()
```



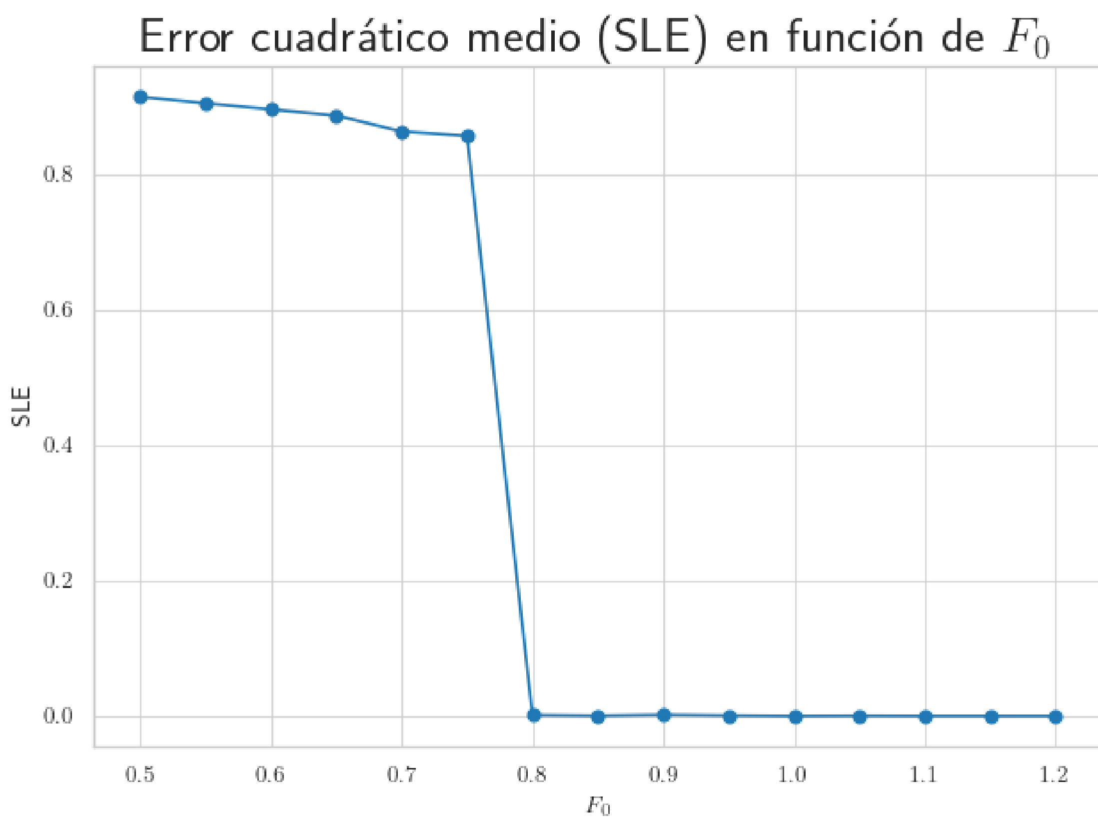
[149]:
```
n = 100
F_init = np.empty(n-2)
z = np.linspace(0, 1, n)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

[150]:
```
error_radio = []
radio = []
for i in range(10, 25):
    F_0 = 0.05*i
    F0 = F_0
    F1 = F_0
    F_init.fill(F_0)

    a_sol, k_sol = fsolve(params_catenaria, initial_guess)
    cat_f = catenaria(z)

    sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
    Famp = np.concatenate(([F0], sol_grad.x, [F1]))
```

```
    error = np.linalg.norm(Famp - cat_f)/np.linalg.norm(cat_f)
    error_radio.append(error)
    radio.append(F_0)
```

```
C:\Users\ismag\AppData\Local\Temp\ipykernel_19584\1344807172.py:9:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last five Jacobian evaluations.
  a_sol, k_sol = fsolve(params_catenaria, initial_guess)
C:\Users\ismag\AppData\Local\Temp\ipykernel_19584\1344807172.py:9:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  a_sol, k_sol = fsolve(params_catenaria, initial_guess)
```

[151]:
```
plt.plot(radio, error_radio, 'o-', label='Gradiente')
# plt.yscale('log')
plt.ylabel('SLE')
plt.xlabel('$F_0$')
plt.title('Error cuadrático medio (SLE) en función de $F_0$', fontsize=20)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
plt.savefig('Figuras/Presentación/error_radios.pdf', format='pdf')
plt.show()
```



Como se observa en la representación gráfica de estas soluciones, todas en las que aumenta el valor de F_0 respecto de la unidad, presentan soluciones similares en cuanto a que se trata de una catenoide ajustada a esos puntos en el inicio y el final del intervalo.

Por otro lado, en los casos en los que F_0 decrece, se presenta un comportamiento anómalo. En valores mayores que $F_0 \approx 0.7$ no se produce ningún cambio de tendencia respecto a lo observado anteriormente. Sin embargo, para condiciones de contorno con radios menores o iguales que ese, la solución calculada presenta un comportamiento que puede dar lugar a pensar que se produce por una falta de precisión del algoritmo de optimización, una necesitad de más puntos u otros problemas derivados de la aplicación numérica de los algoritmos de optimización.

Influencia del número de puntos de la discretización. A continuación, se tratará cómo afecta el número de puntos n en los que se discretiza el intervalo $[0, 1]$, tomando para todos ellos el mismo esquema de partición equiespaciada y manteniendo el mismo valor de las condiciones de contorno y el resto de características del proceso.

[161]:
```
# Recuperamos los valores de F_0
F_0 = 1
F0 = F_0
F1 = F_0
```

- $n = 5$

[162]:
```
n = 5
z5 = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f5 = catenaria(z5)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
time_start = time.time()
sol_gradn5 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()
time_n5 = time_end - time_start
print('Tiempo de ejecución para n=5:', time_n5)
Fampn5 = np.concatenate(([F0], sol_gradn5.x, [F1]))
error_n5 = np.linalg.norm(Fampn5 - cat_f5)/np.linalg.norm(cat_f5)
print('El error para n=5 es:', error_n5)
```

```
Tiempo de ejecución para n=5: 0.018006324768066406
El error para n=5 es: 0.0037659102469243745
```

- $n = 10$

[163]:
```
n = 10
z10 = np.linspace(0, 1, n)
F_init = np.empty(n-2)
```

```
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f10 = catenaria(z10)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
time_start = time.time()
sol_gradn10 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()
time_n10 = time_end - time_start
print('Tiempo de ejecución para n=10:', time_n10)
Fampn10 = np.concatenate(([F0], sol_gradn10.x, [F1]))
error_n10 = np.linalg.norm(Fampn10 - cat_f10)/np.linalg.norm(cat_f10)
print('El error para n=10 es:', error_n10)
```

```
Tiempo de ejecución para n=10: 0.05054640769958496
El error para n=10 es: 0.0019433253695356292
```

- $n = 20$

[164]:
```
n = 20
z20 = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f20 = catenaria(z20)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
time_start = time.time()
sol_gradn20 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()
time_n20 = time_end - time_start
print('Tiempo de ejecución para n=20:', time_n20)
Fampn20 = np.concatenate(([F0], sol_gradn20.x, [F1]))
error_n20 = np.linalg.norm(Fampn20 - cat_f20)/np.linalg.norm(cat_f20)
print('El error para n=20 es:', error_n20)
```

```
Tiempo de ejecución para n=20: 0.1690828800201416
El error para n=20 es: 0.0010218191336893907
```

- $n = 50$

[165]:
```
n = 50
z50 = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f50 = catenaria(z50)
```

```
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
time_start = time.time()
sol_gradn50 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()
time_n50 = time_end - time_start
print('Tiempo de ejecución para n=50:', time_n50)
Fampn50 = np.concatenate(([F0], sol_gradn50.x, [F1]))
error_n50 = np.linalg.norm(Fampn50 - cat_f50)/np.linalg.norm(cat_f50)
print('El error para n=50 es:', error_n50)
```

```
Tiempo de ejecución para n=50: 1.057715892791748
El error para n=50 es: 0.0006467242476269547
```

- $n = 100$

[166]:
```
n = 100
z100 = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f100 = catenaria(z100)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
time_start = time.time()
sol_gradn100 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()
time_n100 = time_end - time_start
print('Tiempo de ejecución para n=100:', time_n100)
Fampn100 = np.concatenate(([F0], sol_gradn100.x, [F1]))
error_n100 = np.linalg.norm(Fampn100 - cat_f100)/np.linalg.norm(cat_f100)
print('El error para n=100 es:', error_n100)
```

```
Tiempo de ejecución para n=100: 6.170040845870972
El error para n=100 es: 0.0003386263379216046
```

- $n = 200$

[167]:
```
n = 200
z200 = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f200 = catenaria(z200)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

```
time_start = time.time()
sol_gradn200 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()
time_n200 = time_end - time_start
print('Tiempo de ejecución para n=200:', time_n200)
Fampn200 = np.concatenate(([F0], sol_gradn200.x, [F1]))
error_n200 = np.linalg.norm(Fampn200 - cat_f200)/np.linalg.norm(cat_f200)
print('El error para n=200 es:', error_n200)
```

```
Tiempo de ejecución para n=200: 38.51359558105469
El error para n=200 es: 0.0013070462813833572
```

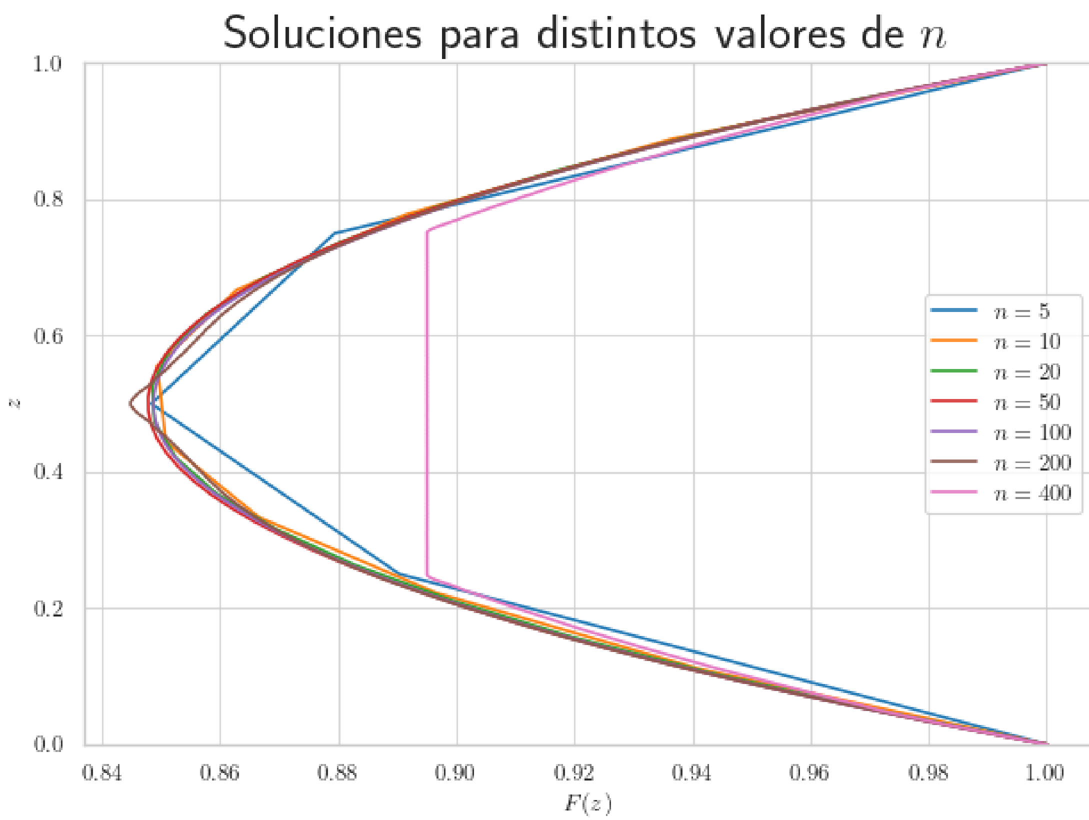
- $n = 400$

[168]:
```
n = 400
z400 = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
cat_f400 = catenaria(z400)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
time_start = time.time()
sol_gradn400 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
time_end = time.time()
time_n400 = time_end - time_start
print('Tiempo de ejecución para n=400:', time_n400)
Fampn400 = np.concatenate(([F0], sol_gradn400.x, [F1]))
error_n400 = np.linalg.norm(Fampn400 - cat_f400)/np.linalg.norm(cat_f400)
print('El error para n=400 es:', error_n400)
```

```
Tiempo de ejecución para n=400: 112.80622887611389
El error para n=400 es: 0.02872582898714192
```

[172]:
```
plt.figure()
plt.plot(Fampn5, z5, label='$n = 5$')
plt.plot(Fampn10, z10, label='$n = 10$')
plt.plot(Fampn20, z20, label='$n = 20$')
plt.plot(Fampn50, z50, label='$n = 50$')
plt.plot(Fampn100, z100, label='$n = 100$')
plt.plot(Fampn200, z200, label='$n = 200$')
plt.plot(Fampn400, z400, label='$n = 400$')
plt.legend()
plt.ylabel('$z$')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
# plt.xlim(0, 1)
plt.title('Soluciones para distintos valores de $n$', fontsize=20)
```

```
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.savefig('Figuras/sol_n.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_n.pdf', format='pdf')
plt.show()
```

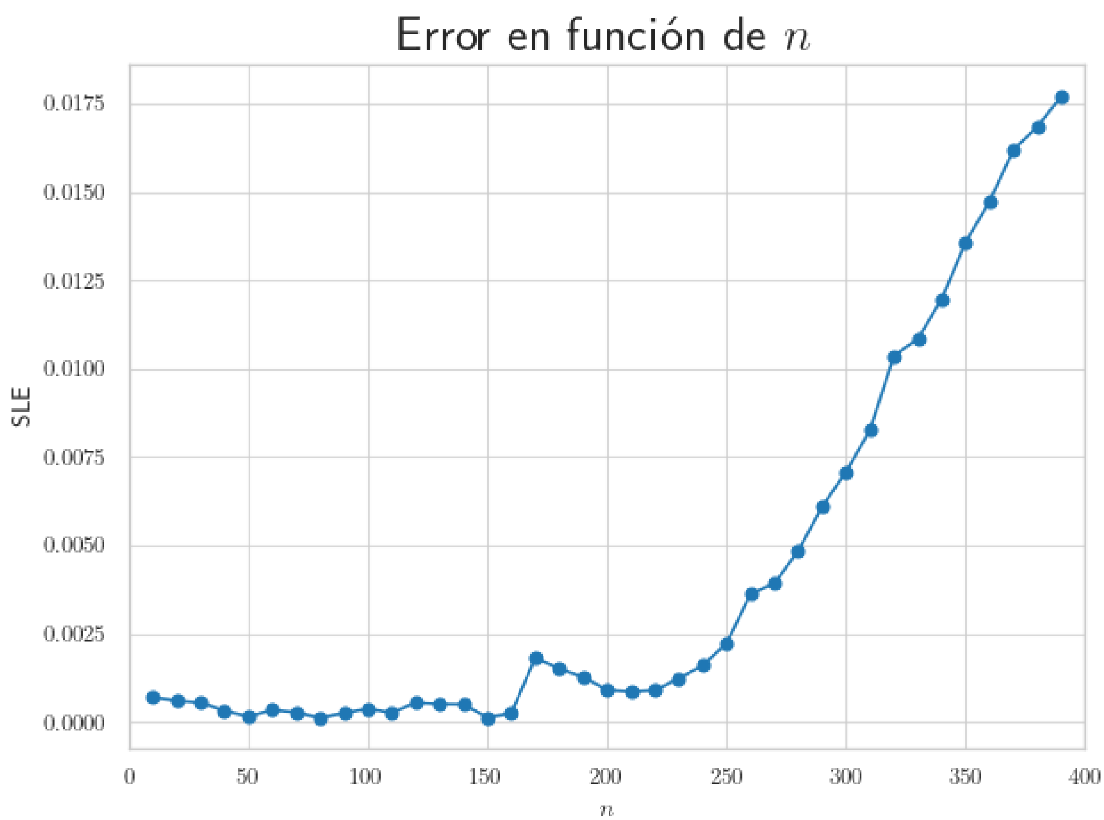


[158]:
```
error_grad_n = []
n_plot = []

for i in range(1, 40):
    n = 10*i
    z = np.linspace(0, 1, n)
    F_init = np.empty(n-2)
    F_init.fill(F_0)
    a_sol, k_sol = fsolve(params_catenaria, initial_guess)
    cat_f = catenaria(z)
    lb = np.zeros(n-2)
    ub = np.ones(n-2) * np.inf
    bounds = np.vstack((lb, ub)).T
    time_start = time.time()
    sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
```

```python
    time_end = time.time()
    time_n = time_end - time_start
    Famp = np.concatenate(([F0], sol_grad.x, [F1]))
    error_n = np.linalg.norm(Famp - cat_f)/np.linalg.norm(cat_f)
    print('El error para n=', n, 'es:', error_n)
    # print('Tiempo de ejecución para n=', n, ':', time_n)
    error_grad_n.append(error_n)
    n_plot.append(n)
```

```
El error para n= 10 es: 0.0007028534613121504
El error para n= 20 es: 0.000609510599204606
El error para n= 30 es: 0.0005514814723878184
El error para n= 40 es: 0.00032188672886263085
El error para n= 50 es: 0.00015656228754340216
El error para n= 60 es: 0.00034102719226899243
El error para n= 70 es: 0.0002769078229150609
El error para n= 80 es: 0.00012061978262244304
El error para n= 90 es: 0.0002588314741030373
El error para n= 100 es: 0.0003783228972769672
El error para n= 110 es: 0.0002655467921645618
El error para n= 120 es: 0.0005500182051622331
El error para n= 130 es: 0.000507973648540647
El error para n= 140 es: 0.0004994575789070481
El error para n= 150 es: 0.000132418214602889
El error para n= 160 es: 0.0002518236497961037
El error para n= 170 es: 0.0018243837862615146
El error para n= 180 es: 0.0015056680444229954
El error para n= 190 es: 0.0012801070705689133
El error para n= 200 es: 0.000909656733142402
El error para n= 210 es: 0.0008597027045333137
El error para n= 220 es: 0.0009021641960239483
El error para n= 230 es: 0.0012323024171287322
El error para n= 240 es: 0.0016070031152654156
El error para n= 250 es: 0.0022249273904817357
El error para n= 260 es: 0.003623391182931877
El error para n= 270 es: 0.003925407206652268
El error para n= 280 es: 0.004845033552514129
El error para n= 290 es: 0.006101312597699711
El error para n= 300 es: 0.007082876123114427
El error para n= 310 es: 0.008260948679365344
El error para n= 320 es: 0.010363969798299025
El error para n= 330 es: 0.010839461539932427
El error para n= 340 es: 0.01197317473718847
El error para n= 350 es: 0.013577428478980589
El error para n= 360 es: 0.01473157602323335
El error para n= 370 es: 0.016193873677862243
El error para n= 380 es: 0.01683808586869473
El error para n= 390 es: 0.017707362257477228
```

[173]:
```python
plt.figure()
plt.plot(n_plot, error_grad_n, 'o-', label='Gradiente')
# plt.xscale('log')
# plt.yscale('log')
plt.xlim(0, 400)
plt.ylabel('SLE')
plt.xlabel('$n$')
plt.title('Error en función de $n$', fontsize=20)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
plt.savefig('Figuras/Presentación/error_n.pdf', format='pdf')
plt.show()
```



A raíz de esta imagen, se pueden analizar los datos dividiéndolos en tres tendencias de comportamiento distintas y que se van a comentar a continuación por separado.

1. **Pocos puntos** (5 o 10 en la imagen): En estas curvas se aprecia el error del cálculo de la solución por el método empleado para el cálculo de la derivada. Este método presenta un error de truncamiento proporcional al tamaño del intervalo escogido entre puntos, y este es mayor cuantos menos puntos se escojan para aproximar la función.
2. **Cantidad moderada de puntos** (entre 20 y 100 en la imagen): Estos esquemas son los que mejor se ajustan a la solución. Apenas se pueden apreciar cambios entre las curvas trazadas

para dichos valores, por lo que se considera un valor de compromiso intermedio a fin de tener suficiente margen en la estabilidad de la solución.

3. **Excesiva cantidad de puntos** (200, 400 o más): En este caso, los errores que se muestran son causa de la aproximación numérica de la derivada, pero esta vez asociada al error de redondeo. Este es el error asociado a la precisión finita del ordenador, proporcional a la constante de Lebesgue Λ_N, que, en el caso de una malla equiespaciada, es $\Lambda_N \geq \frac{2^N\sqrt{2}}{\pi N(N-1)}$, donde N es el número de puntos en los que se divide el intervalo. Es fácil observar que dicha constante puede alcanzar un valor muy alto incluso para valores moderados de N. En el caso que se considera de 200 puntos, la cota inferior es $\Lambda_N \geq 1.8175e + 55$, que, aunque se multiplique por el valor de la precisión numérica de Python (del orden de 10^{-16}), genera unas cotas de error para nada despreciables, pudiendo acarrear errores en la solución como los observados en la gráfica.

Influencia de la distribución de puntos. En este caso, se va a considerar el efecto de considerar diferentes distribuciones de puntos en las que dividir el intervalo del problema.

Una vez más, se consideran el resto de parámetros idénticos al primer caso de estudio, salvo por el número de puntos n, que se va a modificar a fin de apreciar el comportamiento de distintas distribuciones de puntos en los casos más exigentes, es decir, en los casos extremos de n muy grande o muy pequeño. El caso de n intermedio se omite por presentar soluciones casi idénticas en todos los casos.

Caso de n razonablemente pequeño.

[263]:
```
n = 10
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

# planteamos la función que calcula el área tomando delta_z como un array␣
↪definido anteriormente
def area_distrib(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)

    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z[i] # esquema adelantado para el␣
↪cálculo de la derivada
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z[i]

    return integral
```

- Distribución de puntos **equiespaciada**

[264]:
```python
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cateq = catenaria(zeq)
error_eq = np.linalg.norm(Famp_eq - F_cateq)/np.linalg.norm(F_cateq)
print('El error para distribución equiespaciada es:', error_eq)
```

```
El error para distribución equiespaciada es: 0.0019433254436008404
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**: $z_i = \cos\left(\frac{\frac{\pi}{2}+\pi i}{n+1}\right), \quad i = 0, \ldots, n$

[265]:
```python
def chebyshev_zeros(a, b, n):
    cheb_zeros = np.cos((2 * np.arange(1, n+1) - 1) * np.pi / (2 * n))
    mapped_zeros = 0.5 * (b - a) * (cheb_zeros + 1) + a
    sorted_zeros = np.sort(mapped_zeros) # porque los calcula en orden␣
 ↪decreciente
    return sorted_zeros

zchebz = chebyshev_zeros(0, 1, n)
delta_z = np.diff(zchebz)
sol_gradchebz = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebz = catenaria(zchebz)
error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)/np.linalg.norm(F_catchebz)
print('El error para los nodos de Chebyshev es:', error_chebz)
```

```
El error para los nodos de Chebyshev es: 0.005537720056351027
```

- Distribución de puntos según un esquema de **extremos de Chebyshev**: $z_i = \cos\left(\frac{\pi i}{n}\right), \quad i = 0, \ldots, n$

[266]:
```python
def chebyshev_extremes(a, b, n):
    cheb_extremes = np.cos(np.arange(n) * np.pi / (n - 1))
    mapped_extremes = 0.5 * (b - a) * (cheb_extremes + 1) + a
    sorted_extremes = np.sort(mapped_extremes)
    return sorted_extremes

zchebe = chebyshev_extremes(0, 1, n)
delta_z = np.diff(zchebe)
sol_gradchebe = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebe = catenaria(zchebe)
error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)/np.linalg.norm(F_catchebe)
print('El error para los extremos de Chebyshev es:', error_chebe)
```

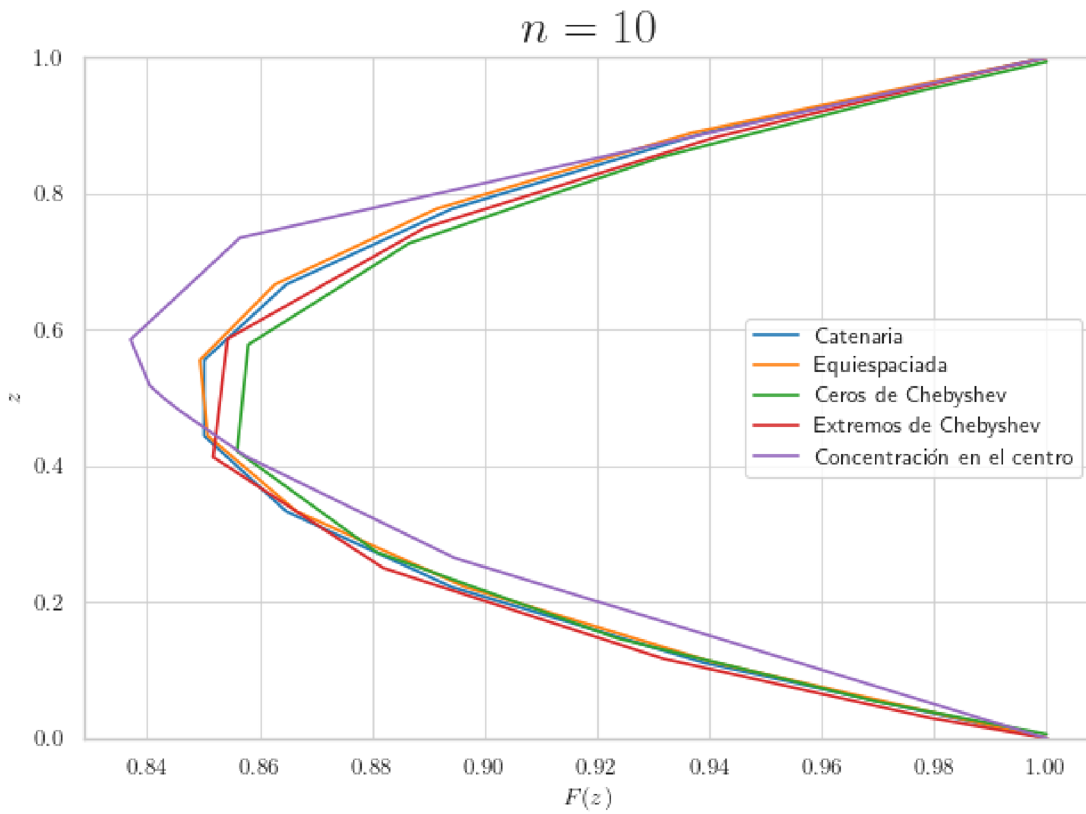
El error para los extremos de Chebyshev es: 0.0031665998435380046

- Distribución que concentra puntos en el centro

[267]:
```
def concentracion_centro(n):
    zconc = np.linspace(-1, 1, n)
    zconc = zconc**3
    zconc = (zconc + 1)/2
    return zconc

zconc = concentracion_centro(n)
delta_z = np.diff(zconc)
sol_gradconc = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_conc = np.concatenate(([F0], sol_gradconc.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catconc = catenaria(zconc)
error_conc = np.linalg.norm(Famp_conc - F_catconc)/np.linalg.norm(F_catconc)
print('El error para concentración en el centro es:', error_conc)
```

El error para concentración en el centro es: 0.012287383722660961

[268]:
```
plt.plot(F_cateq, zeq, label='Catenaria')
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
plt.plot(Famp_conc, zconc, label = 'Concentración en el centro')
plt.ylabel('$z$')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
# plt.xlim(0.8, 1)
# puede ser que distribución concentrada en el centro no mejore porque en␣
↪realidad no hay tanta variación, solo es la escala????
# plt.xlim(0, 1)
plt.title('$n = {}$'.format(n), fontsize=20)
plt.legend()
plt.tight_layout()
# plt.savefig('Figuras/sol_distrib_n10.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_distrib_n10.pdf', format='pdf')
plt.show()
```



En el caso expuesto, se aprecia una solución que, aunque en los extremos del intervalo se parece mucho, no así en la parte central. Esto es debido a que las distribuciones no equiespaciadas acumulan mayor cantidad de puntos en los extremos del intervalo mientras que, en el centro de este, el espaciado entre ellos es mayor, provocando así que la aproximación de la solución sea más imprecisa, por el mismo motivo que se comentó sobre la influencia del número de puntos anteriormente.

Caso de n muy grande.

[269]:
```
n = 200
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

[270]:
```
# planteamos la función que calcula el área tomando delta_z como un array␣
 ↪definido anteriormente
def area_distrib(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)
    # delta_z = 1 / (n - 1)
```

```
    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z[i] # esquema adelantado para el␣
↪cálculo de la derivada
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z[i]

    return integral
```

- Distribución de puntos **equiespaciada**

[271]:
```
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cateq = catenaria(zeq)
error_eq = np.linalg.norm(Famp_eq - F_cateq)
print('El error para distribución equiespaciada es:', error_eq)
```

```
El error para distribución equiespaciada es: 0.01807891226018332
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**

[272]:
```
def chebyshev_zeros(a, b, n):
    cheb_zeros = np.cos((2 * np.arange(1, n+1) - 1) * np.pi / (2 * n))
    mapped_zeros = 0.5 * (b - a) * (cheb_zeros + 1) + a
    sorted_zeros = np.sort(mapped_zeros) # porque los calcula en orden␣
↪decreciente
    return sorted_zeros

zchebz = chebyshev_zeros(0, 1, n)
delta_z = np.diff(zchebz)
sol_gradchebz = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebz = catenaria(zchebz)
error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)
print('El error para los nodos de Chebyshev es:', error_chebz)
```

```
El error para los nodos de Chebyshev es: 0.012021660678577325
```

- Distribución de puntos según un esquema de **extremos de Chebyshev**

[273]:
```
def chebyshev_extremes(a, b, n):
    cheb_extremes = np.cos(np.arange(n) * np.pi / (n - 1))
    mapped_extremes = 0.5 * (b - a) * (cheb_extremes + 1) + a
```

```
    sorted_extremes = np.sort(mapped_extremes)
    return sorted_extremes

zchebe = chebyshev_extremes(0, 1, n)
delta_z = np.diff(zchebe)
sol_gradchebe = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebe = catenaria(zchebe)
error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)
print('El error para los extremos de Chebyshev es:', error_chebe)
```

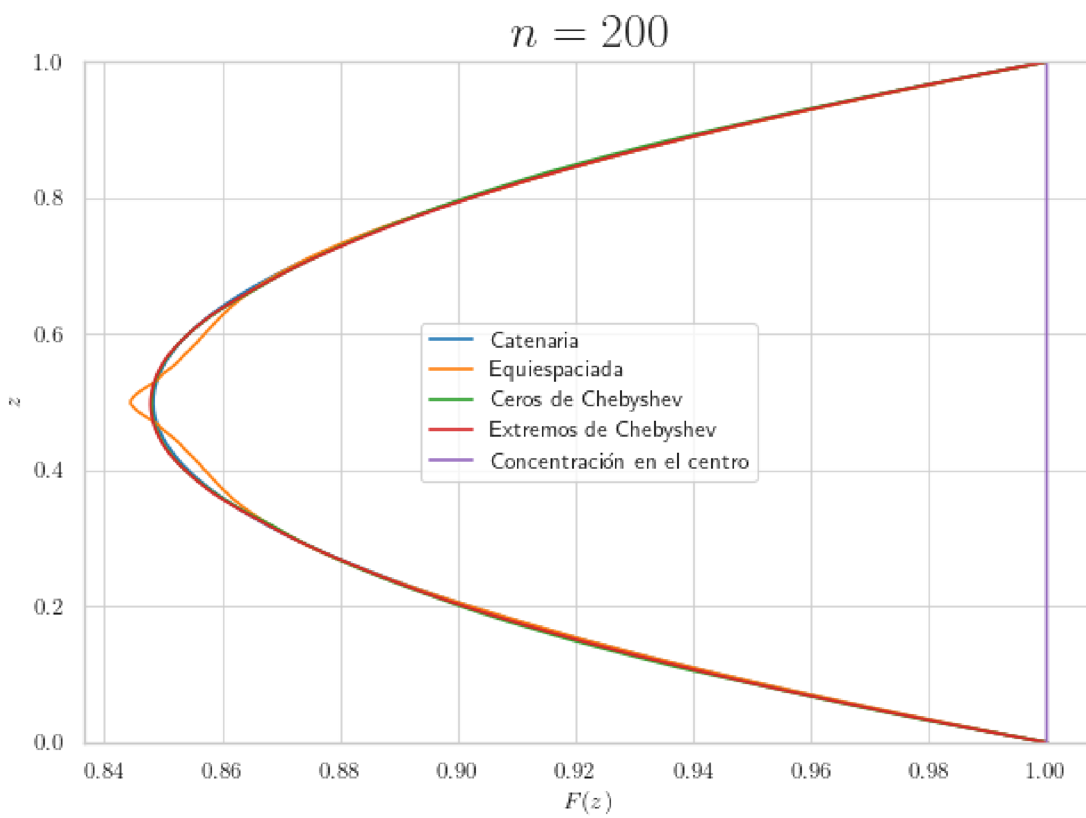
```
El error para los extremos de Chebyshev es: 0.006689913533817554
```

- Distribución que concentra puntos en el centro

[274]:
```
def concentracion_centro(n):
    zconc = np.linspace(-1, 1, n)
    zconc = zconc**3
    zconc = (zconc + 1)/2
    return zconc

zconc = concentracion_centro(n)
delta_z = np.diff(zconc)
sol_gradconc = minimize(area_distrib, F_init, bounds=bounds, method='SLSQP')
Famp_conc = np.concatenate(([F0], sol_gradconc.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catconc = catenaria(zconc)
error_conc = np.linalg.norm(Famp_conc - F_catconc)/np.linalg.norm(F_catconc)
print('El error para concentración en el centro es:', error_conc)
```

```
El error para concentración en el centro es: 0.15470017503913341
```

[275]:
```
plt.plot(F_cateq, zeq, label='Catenaria')
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
plt.plot(Famp_conc, zconc, label = 'Concentración en el centro')
plt.legend()
plt.ylabel('$z$')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
# plt.xlim(0.8, 1)
plt.title('$n = {}$'.format(n), fontsize=20)
plt.tight_layout()
# plt.savefig('Figuras/sol_distrib_n200.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_distrib_n200.pdf', format='pdf')
plt.show()
```



Al contrario que en el caso anterior, ahora las distribuciones que mejor se comportan son las no equiespaciadas. En este caso se debe a que, como se comentó en el apartado sobre la influencia del número de puntos, las distribuciones equiespaciadas presentan un error de redondeo que se dispara al aumentar el número de puntos, que es lo que ocurre en este caso.

Por contra, las distribuciones no equiespaciadas acumulan más puntos en los extremos, que es donde se produce un cambio más pronunciado de las propiedades, consiguiendo adaptarse así mejor, y manteniendo una densidad de puntos moderada en los puntos centrales, reduciendo así el error de redondeo.

Influencia del esquema de cálculo de derivadas. En este caso, se pasará a estudiar la influencia del esquema de derivación en la solución obtenida del problema. Los esquemas de diferencias finitas que se consideran para el cálculo se tratan de los más sencillos, que son los estudiados en la asignatura.

Para ello, en primer lugar, se implementa en el código el cálculo de las derivadas por cada uno de los esquemas que se mencionan a continuación.

- Diferencias finitas progresivas: $\frac{\partial F}{\partial z} = \frac{F(z+\delta_z)-F(z)}{\delta_z}$

[276]:
```
def area_forw(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
```

```python
    n = np.size(Famp)
    delta_z = 1 / (n - 1)

    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z

    return integral
```

- Diferencias finitas regresivas: $\frac{\partial F}{\partial z} = \frac{F(z) - F(z-\delta_z)}{\delta_z}$

[277]:
```python
def area_back(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)
    delta_z = 1 / (n - 1)

    integral = 0
    for i in range(1, n):
        # calcular la derivada
        F_der = (Famp[i] - Famp[i-1]) / delta_z
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z

    return integral
```

- Diferencias finitas centradas: $\frac{\partial F}{\partial z} = \frac{F(z+\delta_z) - F(z-\delta_z)}{2\delta_z}$

[278]:
```python
def area_cent(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)
    delta_z = 1 / (n - 1)

    integral = 0
    for i in range(n):
        if i == 0:  # forward difference at the first point
            F_der = (Famp[i+1] - Famp[i]) / delta_z
        elif i == n-1:  # backward difference at the last point
            F_der = (Famp[i] - Famp[i-1]) / delta_z
        else:  # central difference at interior points
            F_der = (Famp[i+1] - Famp[i-1]) / (2*delta_z)

        integrando = Famp[i] * np.sqrt(1 + F_der**2)
```

```python
        integral += integrando * delta_z

    return integral
```

- Diferencias finitas de paso complejo: $\frac{\partial F}{\partial z} = \frac{\mathrm{Im}(F(z+i\delta_z))}{\delta_z}$

[279]:
```python
h = 1e-8

def area_comp(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)
    delta_z = 1 / (n - 1)

    integral = 0
    for i in range(n-1):
        # Perturb the value with a complex step
        F_complex = Famp.copy().astype(complex)
        F_complex[i] += 1j * h

        # Compute the derivative using the complex step method
        F_der = np.imag(F_complex[i] / h)
        # print(F_complex[i])
        # print('derivada', F_der)

        # Calculate the value of the integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z

    return integral.real
```

A continuación, se va a observar el efecto del esquema de cálculo de la derivada en función del número de puntos escogidos, en todos los casos bajo el mismo esquema de partición del intervalo equiespaciado.

Para un número bajo de puntos: $n = 10$.

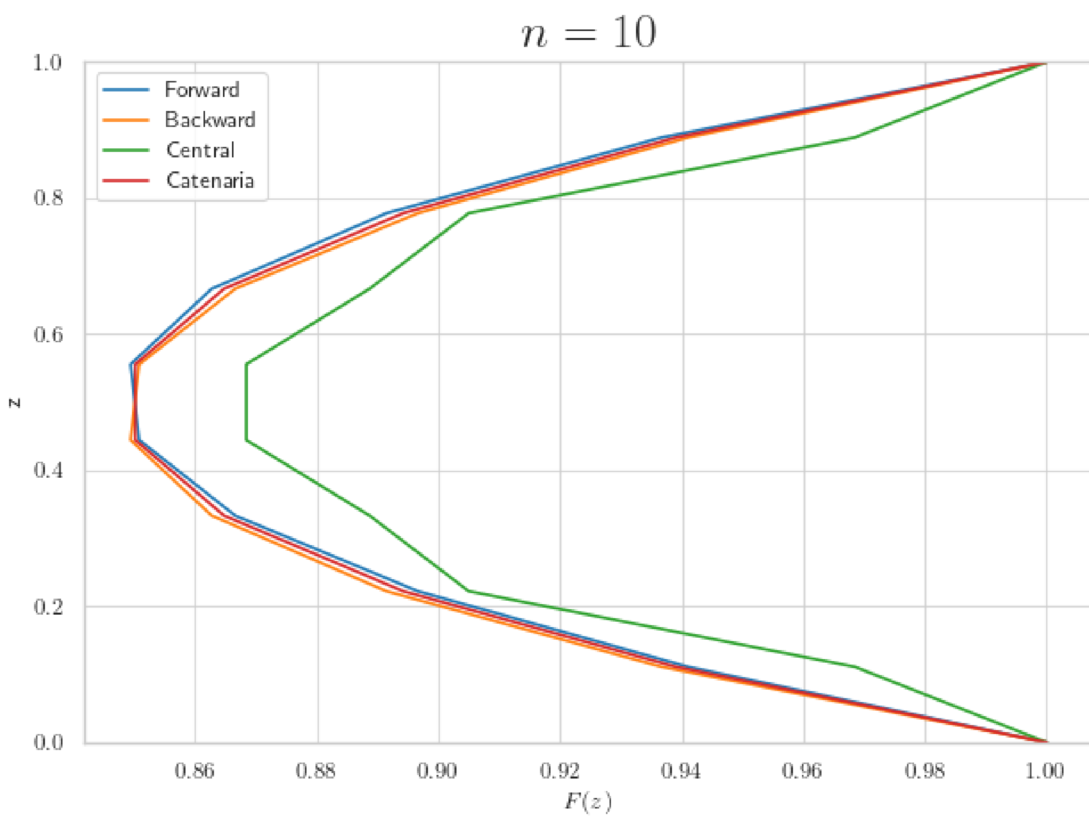
[280]:
```python
n = 10
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)
error_forw = np.array([])
error_back = np.array([])
error_cent = np.array([])
```

```python
error_comp = np.array([])
```

[281]:
```python
sol_gradforw = minimize(area_forw, F_init, bounds=bounds, method='SLSQP')
Famp_forw = np.concatenate(([F0], sol_gradforw.x, [F1]))
error_forw = np.append(error_forw, np.linalg.norm(Famp_forw - Fcat))
print('El error para forward difference es:', error_forw[-1])

sol_gradback = minimize(area_back, F_init, bounds=bounds, method='SLSQP')
Famp_back = np.concatenate(([F0], sol_gradback.x, [F1]))
error_back = np.append(error_back, np.linalg.norm(Famp_back - Fcat))
print('El error para backward difference es:', error_back[-1])

sol_gradcent = minimize(area_cent, F_init, bounds=bounds, method='SLSQP')
Famp_cent = np.concatenate(([F0], sol_gradcent.x, [F1]))
error_cent = np.append(error_cent, np.linalg.norm(Famp_cent - Fcat))
print('El error para central difference es:', error_cent[-1])

sol_gradcomp = minimize(area_comp, F_init, bounds=bounds, method='SLSQP')
Famp_comp = np.concatenate(([F0], sol_gradcomp.x, [F1]))
error_comp = np.append(error_comp, np.linalg.norm(Famp_comp - Fcat))
print('El error para complex step es:', error_comp[-1])
```

```
El error para forward difference es: 0.005600031297052606
El error para backward difference es: 0.005600029631422671
El error para central difference es: 0.06140265490282978
El error para complex step es: 2.5107863952786564
```

[282]:
```python
plt.plot(Famp_forw, z, label = 'Forward')
plt.plot(Famp_back, z, label = 'Backward')
plt.plot(Famp_cent, z, label = 'Central')
plt.plot(Fcat, z, label = 'Catenaria')
# plt.plot(Famp_comp, z, label = 'Complex')
plt.legend()
plt.ylabel('z')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
# plt.xlim(0, 1)
plt.title('$n = {}$'.format(n), fontsize=20)
plt.tight_layout()
# plt.savefig('Figuras/sol_deriv_n10.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_deriv_n10.pdf', format='pdf')
plt.show()
```



Antes de nada, cabe reseñar la proximidad de la solución de las diferencias finitas adelantadas y las retrasadas, tendencia que se va a mantener para todos los casos, por lo que, cuando se hable de diferencias finitas descentradas, se hará referencia a ambas en lo que sigue, para evitar repeticiones innecesarias.

En este caso, se puede apreciar como las diferencias finitas descentradas generan un mejor resultado en la aproximación de la solución. Esto es debido a que el paso en la fórmula de las diferencias finitas centradas es el doble que en las otras, por lo que se procuce un cambio más pronunciado en las propiedades en ese espaciado que en las descentradas, acarreando así mayor error en la aproximación de la función.

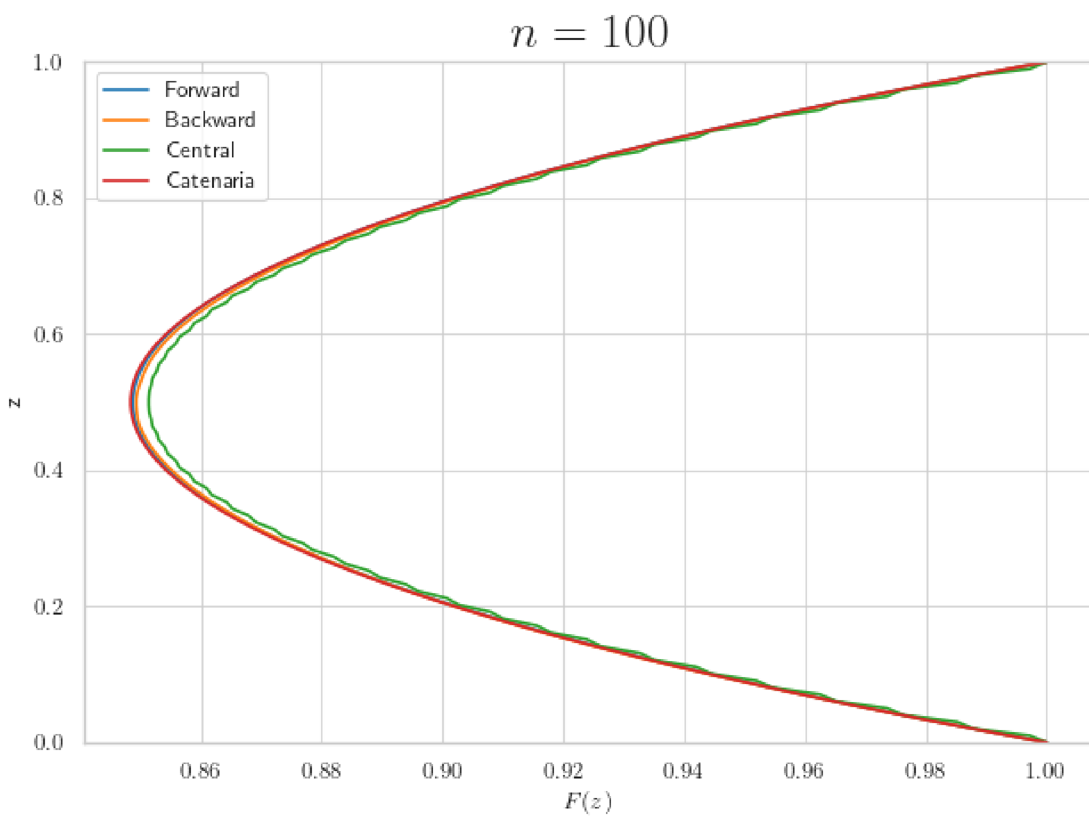
Para un número moderado de puntos: $n = 100$.

[283]:
```
n = 100
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)
```

[284]:
```
sol_gradforw = minimize(area_forw, F_init, bounds=bounds, method='SLSQP')
Famp_forw = np.concatenate(([F0], sol_gradforw.x, [F1]))
error_forw = np.append(error_forw, np.linalg.norm(Famp_forw - Fcat))
print('El error para forward difference es:', error_forw[-1])

sol_gradback = minimize(area_back, F_init, bounds=bounds, method='SLSQP')
Famp_back = np.concatenate(([F0], sol_gradback.x, [F1]))
error_back = np.append(error_back, np.linalg.norm(Famp_back - Fcat))
print('El error para backward difference es:', error_back[-1])

sol_gradcent = minimize(area_cent, F_init, bounds=bounds, method='SLSQP')
Famp_cent = np.concatenate(([F0], sol_gradcent.x, [F1]))
error_cent = np.append(error_cent, np.linalg.norm(Famp_cent - Fcat))
print('El error para central difference es:', error_cent[-1])

sol_gradcomp = minimize(area_comp, F_init, bounds=bounds, method='SLSQP')
Famp_comp = np.concatenate(([F0], sol_gradcomp.x, [F1]))
error_comp = np.append(error_comp, np.linalg.norm(Famp_comp - Fcat))
print('El error para complex step es:', error_comp[-1])
```

```
El error para forward difference es: 0.003049379653914534
El error para backward difference es: 0.006877680601583818
El error para central difference es: 0.025919439084958595
El error para complex step es: 8.893405235777724
```

[285]:
```
plt.plot(Famp_forw, z, label = 'Forward')
plt.plot(Famp_back, z, label = 'Backward')
plt.plot(Famp_cent, z, label = 'Central')
plt.plot(Fcat, z, label = 'Catenaria')
# plt.plot(Famp_comp, z, label = 'Complex')
plt.legend()
plt.ylabel('z')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
# plt.xlim(0, 1)
plt.title('$n = {}$'.format(n), fontsize=20)
plt.tight_layout()
# plt.savefig('Figuras/sol_deriv_n100.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_deriv_n100.pdf', format='pdf')
plt.show()
```



En el caso de escoger una cantidad moderada de puntos, se observa un comportamiento similar al estudiado en el caso anterior. Sin embargo, en este caso, las diferencias finitas centradas se acercan aún más al resultado de las fórmulas descentradas. Esto se produce porque, al aumentar tanto el espaciado, aunque el paso sea mayor que en el caso de las fórmulas descentradas, presenta una aproximación cada vez más aceptable de la solución.

Para un número elevado de puntos: $n = 200$**.**

[286]:
```
n = 200
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)
```

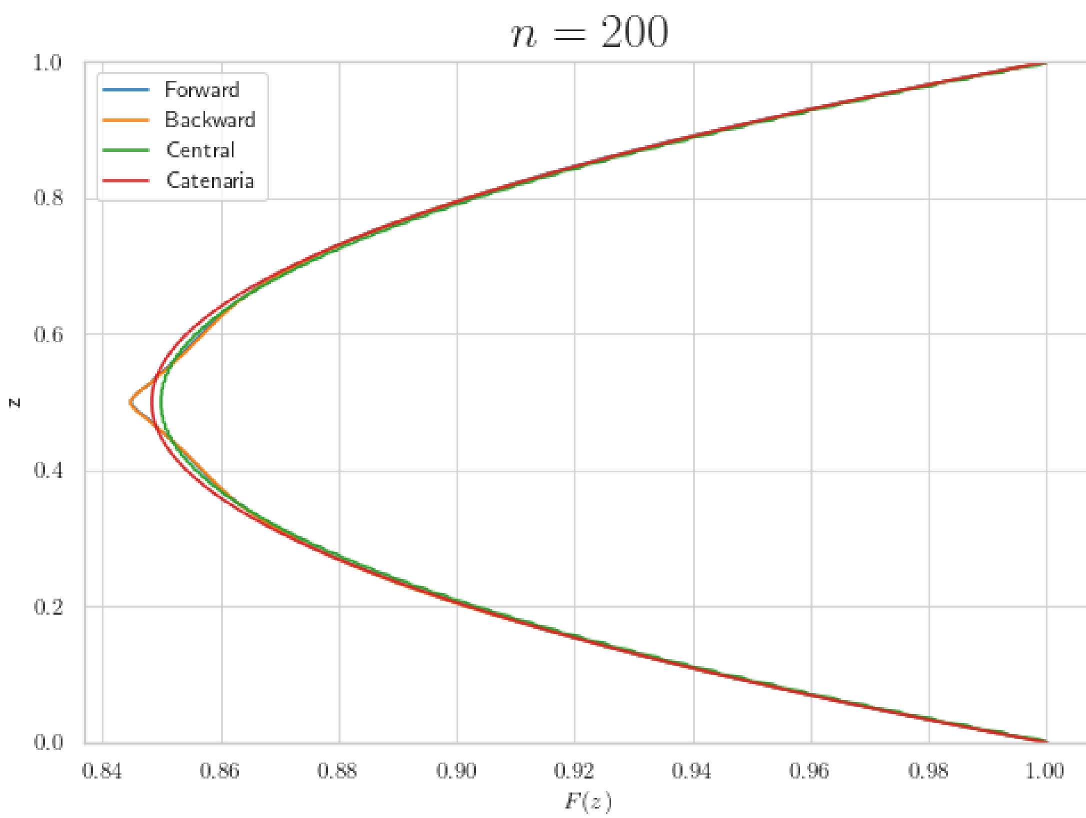
[287]:
```
sol_gradforw = minimize(area_forw, F_init, bounds=bounds, method='SLSQP')
Famp_forw = np.concatenate(([F0], sol_gradforw.x, [F1]))
error_forw = np.append(error_forw, np.linalg.norm(Famp_forw - Fcat))
print('El error para forward difference es:', error_forw[-1])
```

```python
sol_gradback = minimize(area_back, F_init, bounds=bounds, method='SLSQP')
Famp_back = np.concatenate(([F0], sol_gradback.x, [F1]))
error_back = np.append(error_back, np.linalg.norm(Famp_back - Fcat))
print('El error para backward difference es:', error_back[-1])

sol_gradcent = minimize(area_cent, F_init, bounds=bounds, method='SLSQP')
Famp_cent = np.concatenate(([F0], sol_gradcent.x, [F1]))
error_cent = np.append(error_cent, np.linalg.norm(Famp_cent - Fcat))
print('El error para central difference es:', error_cent[-1])

sol_gradcomp = minimize(area_comp, F_init, bounds=bounds, method='SLSQP')
Famp_comp = np.concatenate(([F0], sol_gradcomp.x, [F1]))
error_comp = np.append(error_comp, np.linalg.norm(Famp_comp - Fcat))
print('El error para complex step es:', error_comp[-1])
```

```
El error para forward difference es: 0.016635528865998166
El error para backward difference es: 0.01807696034007079
El error para central difference es: 0.020677694900428326
El error para complex step es: 12.648760871204432
```

[288]:

```python
plt.plot(Famp_forw, z, label = 'Forward')
plt.plot(Famp_back, z, label = 'Backward')
plt.plot(Famp_cent, z, label = 'Central')
plt.plot(Fcat, z, label = 'Catenaria')
# plt.plot(Famp_comp, z, label = 'Complex')
plt.legend()
plt.ylabel('z')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
# plt.xlim(0, 1)
plt.title('$n = {}$'.format(n), fontsize=20)
plt.tight_layout()
# plt.savefig('Figuras/sol_deriv_n200.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_deriv_n200.pdf', format='pdf')
plt.show()
```



Al contrario que en los casos estudiados para menor cantidad de puntos, al aumentar lo suficiente n se produce un cambio de tendencia.

En este caso, ningún esquema que proporciona un gran resultado, aunque el que emplea un cálculo de la derivada siguiendo un esquema de diferencias finitas centradas mantiene más fidelidad con la solución. Precisamente, el cambio respecto a lo considerado anteriormente se debe al mismo motivo por el que antes no se conseguía una aprozimación tan buena: que el paso sea el doble que en las diferencias finitas descentradas.

Al crecer tanto el error de redondeo (según la constante de Lebesge, Λ), las diferencias finitas descentradas para una malla equiespaciada dejan de ser una opción viable en cuanto aumenta el número de puntos por encima de un umbral relativamente bajo. Es así que, para mallas con muchos puntos, se recomienda cambiar el esquema a uno de derivadas finitas centradas.

La cantidad de puntos tendiendo a infinito: $n = 600$.

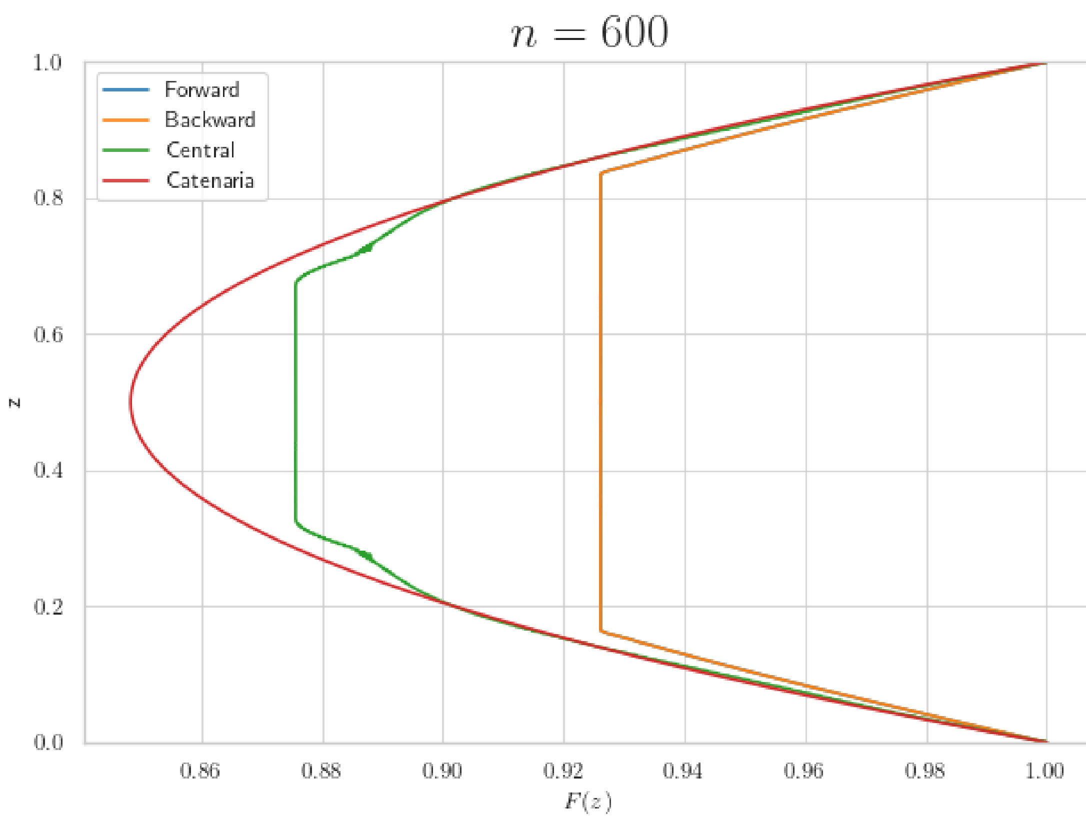
[289]:
```
n = 600
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

```
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)
```

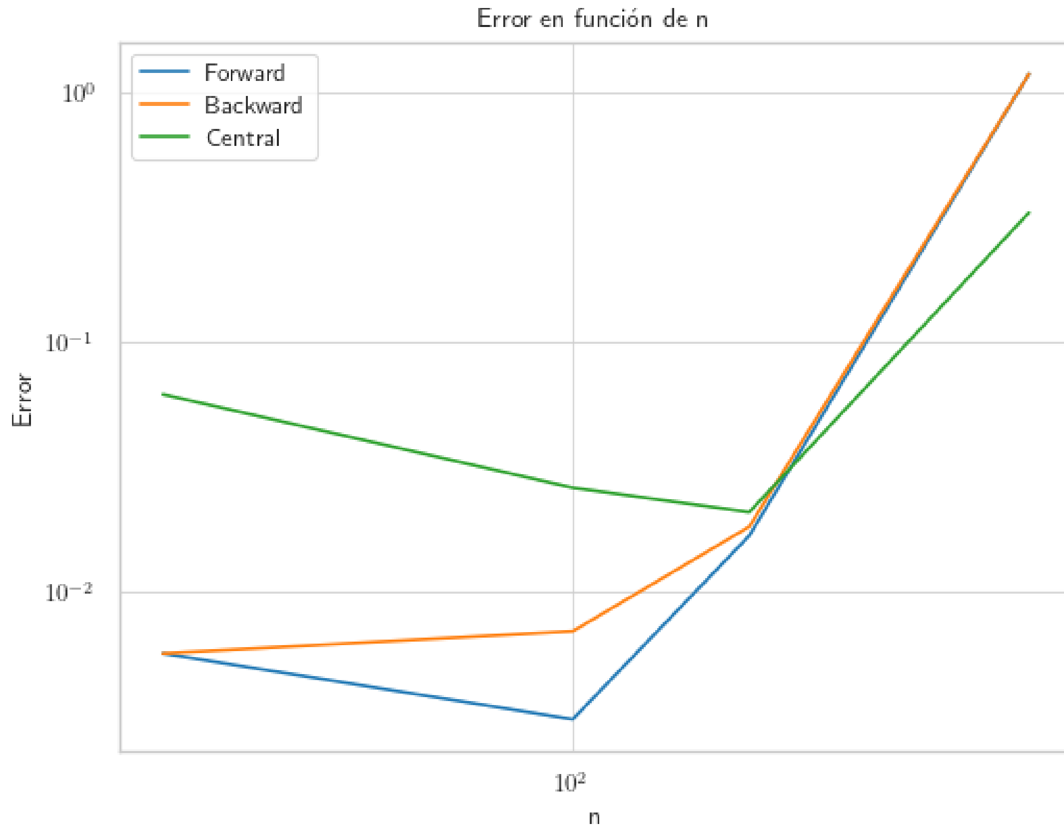
[290]:
```
sol_gradforw = minimize(area_forw, F_init, bounds=bounds, method='SLSQP')
Famp_forw = np.concatenate(([F0], sol_gradforw.x, [F1]))
error_forw = np.append(error_forw, np.linalg.norm(Famp_forw - Fcat))
print('El error para forward difference es:', error_forw[-1])

sol_gradback = minimize(area_back, F_init, bounds=bounds, method='SLSQP')
Famp_back = np.concatenate(([F0], sol_gradback.x, [F1]))
error_back = np.append(error_back, np.linalg.norm(Famp_back - Fcat))
print('El error para backward difference es:', error_back[-1])

sol_gradcent = minimize(area_cent, F_init, bounds=bounds, method='SLSQP')
Famp_cent = np.concatenate(([F0], sol_gradcent.x, [F1]))
error_cent = np.append(error_cent, np.linalg.norm(Famp_cent - Fcat))
print('El error para central difference es:', error_cent[-1])

sol_gradcomp = minimize(area_comp, F_init, bounds=bounds, method='SLSQP')
Famp_comp = np.concatenate(([F0], sol_gradcomp.x, [F1]))
error_comp = np.append(error_comp, np.linalg.norm(Famp_comp - Fcat))
print('El error para complex step es:', error_comp[-1])
```

```
El error para forward difference es: 1.186650823024162
El error para backward difference es: 1.1866235576945554
El error para central difference es: 0.32935354938310174
El error para complex step es: 21.990649304551926
```

[291]:
```
plt.plot(Famp_forw, z, label = 'Forward')
plt.plot(Famp_back, z, label = 'Backward')
plt.plot(Famp_cent, z, label = 'Central')
plt.plot(Fcat, z, label = 'Catenaria')
# plt.plot(Famp_comp, z, label = 'Complex')
plt.legend()
plt.ylabel('z')
plt.xlabel('$F(z)$')
plt.ylim(0, 1)
# plt.xlim(0, 1)
plt.title('$n = {}$'.format(n), fontsize=20)
plt.tight_layout()
# plt.savefig('Figuras/sol_deriv_n600.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_deriv_n600.pdf', format='pdf')
plt.show()
```



[292]:
```
plt.plot([20, 100, 200, 600], error_forw, label = 'Forward')
plt.plot([20, 100, 200, 600], error_back, label = 'Backward')
plt.plot([20, 100, 200, 600], error_cent, label = 'Central')
# plt.plot([20, 100, 200, 600], error_comp, label = 'Complex')
plt.legend()
plt.yscale('log')
plt.xscale('log')
plt.ylabel('Error')
plt.xlabel('n')
plt.title('Error en función de n')
plt.grid(True)
plt.show()
```



Aunque en este caso se consideren "tan solo" 600 puntos, es un buen indicador del efecto que tiene aumentar la cantidad de puntos por encima de un límite admisible, o lo que es lo mismo, en el límite de $n \to \infty$.

Aquí, se observa que sendos esquemas descentrados y centrados sufren errores, una vez más debidos al error de redondeo, a su vez proporcional a la constante de Lebesgue. En este caso, tanto el esquema centrado de cálculo de diferencias finitas como, evidentemente, los esquemas descentrados, presentan amplitudes de los intervalos tan pequeñas entre los puntos que se evalúa la función que el error numérico al dividir por valores tan pequeños se dispara, provocando estas soluciones que para nada se parecen a la solución real del problema.

Influencia del algoritmo de optimización. SEGURAMENTE SEA MEJOR QUITARLO En este apartado, se van a comentar los efectos de los algoritmos de optimización empleados, especialmente en lo que a eficiencia de iteraciones y tiempo de ejecución se refiere. Para ello, se va a

Se calculan tanto el gradiente como la hessiana para los métodos que lo requieren:

[ ]:
```
from autograd import grad
area_func_grad = grad(area_func)
```

```
from autograd import hessian
area_func_hessian = hessian(area_func)
```

Funciones con pocas variables de salida: $n = 20$.

[ ]:
```
n = 20
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)

error_CG = np.array([])
error_BFGS = np.array([])
error_Newton_CG = np.array([])
error_LBFGSB = np.array([])
error_SLSQP = np.array([])
error_trustncg = np.array([])
error_cobyla = np.array([])
error_nelder = np.array([])

time_CG = np.array([])
time_BFGS = np.array([])
time_Newton_CG = np.array([])
time_LBFGSB = np.array([])
time_SLSQP = np.array([])
time_trustncg = np.array([])
time_cobyla = np.array([])
time_nelder = np.array([])
```

1. **Método del *Steepest Descent* (descenso más pronunciado)**

El método del *Steepest Descent* es un algoritmo de optimización que busca minimizar una función objetivo $f(x)$ moviéndose iterativamente en la dirección opuesta al gradiente de la función en el punto actual, ya que le gradiente apunta en la dirección de máximo intemento de la función. Al moverse en la dirección opuesta, se intenta encontrar un punto donde la función tenga un valor mínimo.

Como no se han localizado librerías en las que se encuentre programado el método de *Steepest Descent*, se recoge a continuación el código con el que se ha programado dicho método.

[ ]:
```
class SteepestDescent:
    def __init__(self, area_func, x_init, learning_rate=0.01, max_iters=100000,␣
 ↪epsilon=1e-6):
        self.area_func = area_func
        self.x_init = x_init
```

```python
        self.learning_rate = learning_rate
        self.max_iters = max_iters
        self.epsilon = epsilon
        self.x = None
        self.nit = None
        self.message = None
        self.success = None
        self.status = None
        self.fun = None
        self.nfev = None
        # self.maxcv = None

    def run(self):
        x = self.x_init
        for i in range(self.max_iters):
            grad = area_func_grad(x)
            x_new = x - self.learning_rate * grad  # Actualizar x
            if anp.abs(self.area_func(x_new) - self.area_func(x)) < self.epsilon:
↪  # Si la mejora es muy pequeña, detenerse
                self.message = "Optimization terminated successfully."
                self.success = True
                self.status = 0
                self.fun = self.area_func(x_new)
                self.nfev = i+1
                # self.maxcv = 0.0
                break
            x = x_new
            self.nit = i+1
        else:  # Se ejecuta si el bucle for se agota (se alcanza el número␣
↪máximo de iteraciones)
            self.message = "Maximum number of function evaluations has been␣
↪exceeded."
            self.success = False
            self.status = 2
            self.fun = self.area_func(x_new)
            self.nfev = self.max_iters
            # self.maxcv = 0.0
        self.x = x
```

[ ]:
```python
# time_start = time.time()

# optimizer_steepest = SteepestDescent(area_func, F_init)
# optimizer_steepest.run()

# time_end = time.time()
# time_steepest = time_end - time_start
# Famp_steepest = np.concatenate(([F0], optimizer_steepest.x, [F1]))
```

[ ]:
```python
# print(optimizer_steepest.message)
# print('Valor de la función:', optimizer_steepest.fun)
# # print('Resultado:', optimizer_steepest.x)
# print('Evaluaciones de la función:', optimizer_steepest.nfev)
# print('Tiempo de ejecución:', time_steepest)
```

2. ***Conjugate Gradient* (gradiente conjugado).**

El método del *Conjugate Gradient* es un algoritmo iterativo que combina las ventajas del *Steepest Descent* y los métodos directos, como la factorización LU, para encontrar soluciones eficientes a problemas de optimización.

A diferencia del *Steepest Descent*, que puede zigzaguear, el método del *Conjugate Gradient* genera una serie de direcciones conjugadas que son ortogonales entre sí respecto a la matriz del problema. Esto permite una convergencia más rápida.

Como ventajas respecto de su eficiencia, no requiere almacenamiento de matrices grandes ni operaciones de factorización costosas, lo que lo hace adecuado para problemas de gran escala.

En la imagen siguiente, se aprecia la comparativa entre el método del *Steepest Descent* (en verde) y el método del Gradiente Conjugado (en rojo).

[ ]:
```python
time_start = time.time()

optimizer_CG = minimize(area_func, F_init, method='CG', options={'maxiter':␣
↪100000}) # no se pueden imponer restricciones

time_end = time.time()
time_CG = np.append(time_CG, time_end - time_start)
Famp_CG = np.concatenate(([F0], optimizer_CG.x, [F1]))
error_CG = np.append(error_CG, np.linalg.norm(Famp_CG - Fcat))
```

[ ]:
```python
print(optimizer_CG.message)
print('Valor de la función:', optimizer_CG.fun)
# print('Resultado:', optimizer_CG.x)
print('Evaluaciones de la función:', optimizer_CG.nfev)
print('Tiempo de ejecución:', time_CG[-1])
print('Evaluaciones del gradiente:', optimizer_CG.njev)
print('El error para el método de Newton-CG es:', error_CG[-1])
```

```
Optimization terminated successfully.
Valor de la función: 0.9537486906979558
Evaluaciones de la función: 2318
Tiempo de ejecución: 0.6020419597625732
Evaluaciones del gradiente: 122
El error para el método de Newton-CG es: 0.0038505408710451717
```

3. **Método de Newton de gradiente conjugado.**

El método de Newton del gradiente conjugado es un algoritmo de optimización que combina los principio del método de Newton y del gradiente conjugado para minimizar funciones. Utiliza la información de la matriz Hessiana para mejorar la dirección de búsqueda y la convergencia en comparación con métodos que utilizan únicamente en la primera derivada.

[ ]:
```
time_start = time.time()

optimizer_Newton_CG = minimize(area_func, F_init, method='Newton-CG',␣
↪jac=area_func_grad, hess=area_func_hessian, options={'maxiter': 100000})

time_end = time.time()
time_Newton_CG = np.append(time_Newton_CG, time_end - time_start)
Famp_Newton_CG = np.concatenate(([F0], optimizer_Newton_CG.x, [F1]))
error_Newton_CG = np.append(error_Newton_CG, np.linalg.norm(Famp_Newton_CG -␣
↪Fcat))
```

[ ]:
```
print(optimizer_Newton_CG.message)
print('Valor de la función:', optimizer_Newton_CG.fun)
# print('Resultado:', optimizer_Newton_CG.x)
print('Evaluaciones de la función:', optimizer_Newton_CG.nfev)
print('Tiempo de ejecución:', time_Newton_CG[-1])
print('Evaluaciones del gradiente:', optimizer_Newton_CG.njev)
print('El error para el método de Newton-CG es:', error_Newton_CG[-1])
```

```
Optimization terminated successfully.
Valor de la función: 0.953748690718871
Evaluaciones de la función: 7
Tiempo de ejecución: 1.1240878105163574
Evaluaciones del gradiente: 7
El error para el método de Newton-CG es: 0.003850769917892317
```

4. **Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS)** (quasi-Newton).

El algoritmo BFGS es un método cuasi-Newton que actualiza una aproximación de la inversa de la matriz Hessiana utilizando información del gradiente. En vez de calcular la Hessiana completa, el algoritmo utliliza una serie de actualizaciones para aproximar la inversa de dicha matriz, reduciendo significativamente el costo computacional.

Como todos los métodos cuasi-Newton, el BFGS genera una aproximación de la inversa de la Hessiana, en lugar de calcularla directamente. Es uno de los métodos más robustos y con buena tasa de convergencia, por lo que es uno de los métodos más usados en la actualidad.

[ ]:
```
time_start = time.time()

optimizer_BFGS = minimize(area_func, F_init, method='BFGS', jac=area_func_grad,␣
↪options={'maxiter': 100000})

time_end = time.time()
time_BFGS = np.append(time_BFGS, time_end - time_start)
Famp_BFGS = np.concatenate(([F0], optimizer_BFGS.x, [F1]))
error_BFGS = np.append(error_BFGS, np.linalg.norm(Famp_BFGS - Fcat))
```

[ ]:
```python
print(optimizer_BFGS.message)
print('Valor de la función:', optimizer_BFGS.fun)
# print('Resultado:', optimizer_BFGS.x)
print('Evaluaciones de la función:', optimizer_BFGS.nfev)
print('Tiempo de ejecución:', time_BFGS[-1])
print('Evaluaciones del gradiente:', optimizer_BFGS.njev)
print('El error para el método de BFGS es:', error_BFGS[-1])
```

```
Optimization terminated successfully.
Valor de la función: 0.9537486906273623
Evaluaciones de la función: 28
Tiempo de ejecución: 0.6330444812774658
Evaluaciones del gradiente: 28
El error para el método de BFGS es: 0.003848975593082678
```

5. ***Limited-memory BFGS with Box constraints*** **(L-BFGS-B)**.

Este método combina las características del método L-BFGS (una dle BFGS que utiliza mucha menos memoria) con la capacidad de manejar restricciones simples en sus parámetros, es decir, del tipo $l_b < x < u_b$, donde l_b y u_b son los límites inferior y superior respectivamente.

[ ]:
```python
time_start = time.time()

sol_LBFGSB = minimize(area_func, F_init, method='L-BFGS-B', bounds=bounds,␣
↪options={'maxiter': 100000})

time_end = time.time()
time_LBFGSB = np.append(time_LBFGSB, time_end - time_start)
Famp_LBFGSB = np.concatenate(([F0], sol_LBFGSB.x, [F1]))
error_LBFGSB = np.append(error_LBFGSB, np.linalg.norm(Famp_LBFGSB - Fcat))
```

[ ]:
```python
print(sol_LBFGSB.message)
print('Valor de la función:', sol_LBFGSB.fun)
# print('Resultado:', sol_LBFGSB.x)
print('Evaluaciones de la función:', sol_LBFGSB.nfev)
print('Tiempo de ejecución:', time_LBFGSB[-1])
print('Evaluaciones del gradiente:', sol_LBFGSB.njev)
print('El error para el método de L-BFGS-B es:', error_LBFGSB[-1])
```

```
CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
Valor de la función: 0.9537486917897874
Evaluaciones de la función: 437
Tiempo de ejecución: 0.16801214218139648
Evaluaciones del gradiente: 23
El error para el método de L-BFGS-B es: 0.0038422634420564534
```

6. ***Newton Conjugate Gradient Trust-Region***.

Este es un método que combina los principios del método de Newton, del Gradiente Conjugado y el concepto de las regiones de confianza para minimizar una función.

El único concepto nuevo es el de regiones de confianza, y este se trata de una región alrededor del punto en el que se encuentra la función al inicio de cada iteración, en la que se busca minimizar la función en ese mismo paso.

[ ]:
```
time_start = time.time()

sol_trustncg = minimize(area_func, F_init, method='trust-ncg',␣
↪jac=area_func_grad, hess=area_func_hessian, options={'maxiter': 100000})

time_end = time.time()
time_trustncg = np.append(time_trustncg, time_end - time_start)
Famp_trustncg = np.concatenate(([F0], sol_trustncg.x, [F1]))
error_trustncg = np.append(error_trustncg, np.linalg.norm(Famp_trustncg - Fcat))
```

[ ]:
```
print(sol_trustncg.message)
print('Valor de la función:', sol_trustncg.fun)
# print('Resultado:', sol_trustncg.x)
print('Evaluaciones de la función:', sol_trustncg.nfev)
print('Tiempo de ejecución:', time_trustncg[-1])
print('Evaluaciones del gradiente:', sol_trustncg.njev)
print('El error para el método de trust-ncg es:', error_trustncg[-1])
```

```
Optimization terminated successfully.
Valor de la función: 0.9537486906222621
Evaluaciones de la función: 6
Tiempo de ejecución: 1.8471407890319824
Evaluaciones del gradiente: 6
El error para el método de trust-ncg es: 0.0038487002008285526
```

7. ***Sequential Least Squares Programming*** **(SLSQP)**.

Este algorimo busca minimizar la función construyendo una aprozimación cuadrática local de la función objetivo, generalmente que tiene una forma no lineal. El algoritmo minimiza la aproximación en cada paso, para repetir posteriormente el proceso hasta que converja.

[ ]:
```
start = time.time()

sol_SLSQP = minimize(area_func, F_init, method='SLSQP', bounds=bounds,␣
↪options={'maxiter': 100000})

end = time.time()
time_SLSQP = np.append(time_SLSQP, end - start)
Famp_SLSQP = np.concatenate(([F0], sol_SLSQP.x, [F1]))
error_SLSQP = np.append(error_SLSQP, np.linalg.norm(Famp_SLSQP - Fcat))
```

[ ]:
```
print(sol_SLSQP.message)
print('Valor de la función:', sol_SLSQP.fun)
# print('Resultado:', sol_SLSQP.x)
print('Evaluaciones de la función:', sol_SLSQP.nfev)
```

```python
print('Tiempo de ejecución:', time_SLSQP[-1])
print('Evaluaciones del gradiente:', sol_SLSQP.njev)
print('El error para el método de SLSQP es:', error_SLSQP[-1])
```

```
Optimization terminated successfully
Valor de la función: 0.9537487850868308
Evaluaciones de la función: 429
Tiempo de ejecución: 0.21001791954040527
Evaluaciones del gradiente: 21
El error para el método de SLSQP es: 0.004135763067604287
```

8. **Nelder-Mead**.

Este es un método de búsqueda directa que utiliza un *simplex local* que se va moviendo y adaptando, mediante reflexiones contracciones y expansiones, para ir acercándose al mínimo. Es especialmente útil para la optimización de funciones en las que la evaluación de la derivada resulta costosa, ya que este método no requiere de su cálculo para funcionar.

```python
[ ]: time_start = time.time()

sol_nelder = minimize(area_func, F_init, method='Nelder-Mead', bounds=bounds,␣
 ↪options={'maxiter': 100000})

time_end = time.time()
time_nelder = np.append(time_nelder, time_end - time_start)
Famp_nelder = np.concatenate(([F0], sol_nelder.x, [F1]))
error_nelder = np.append(error_nelder, np.linalg.norm(Famp_nelder - Fcat))
```

```python
[ ]: print(sol_nelder.message)
print('Valor de la función:', sol_nelder.fun)
# print('Resultado:', sol_nelder.x)
print('Evaluaciones de la función:', sol_nelder.nfev)
print('Tiempo de ejecución:', time_nelder[-1])
print('El error para el método de Nelder-Mead es:', error_nelder[-1])
```

```
Optimization terminated successfully.
Valor de la función: 0.9537488716427959
Evaluaciones de la función: 2143
Tiempo de ejecución: 0.6900506019592285
El error para el método de Nelder-Mead es: 0.004045723577738655
```

9. ***Constrained Optimization BY Linear Approximation* (COBYLA)**.

En esencia, el algoritmo COBYLA genera una aproximación lineal de la función en cada iteración que resuelve para acercarse al mínimo de la función. Al igual que el algoritmo anterior, este no necesita calcular ningún tipo de derivadas.

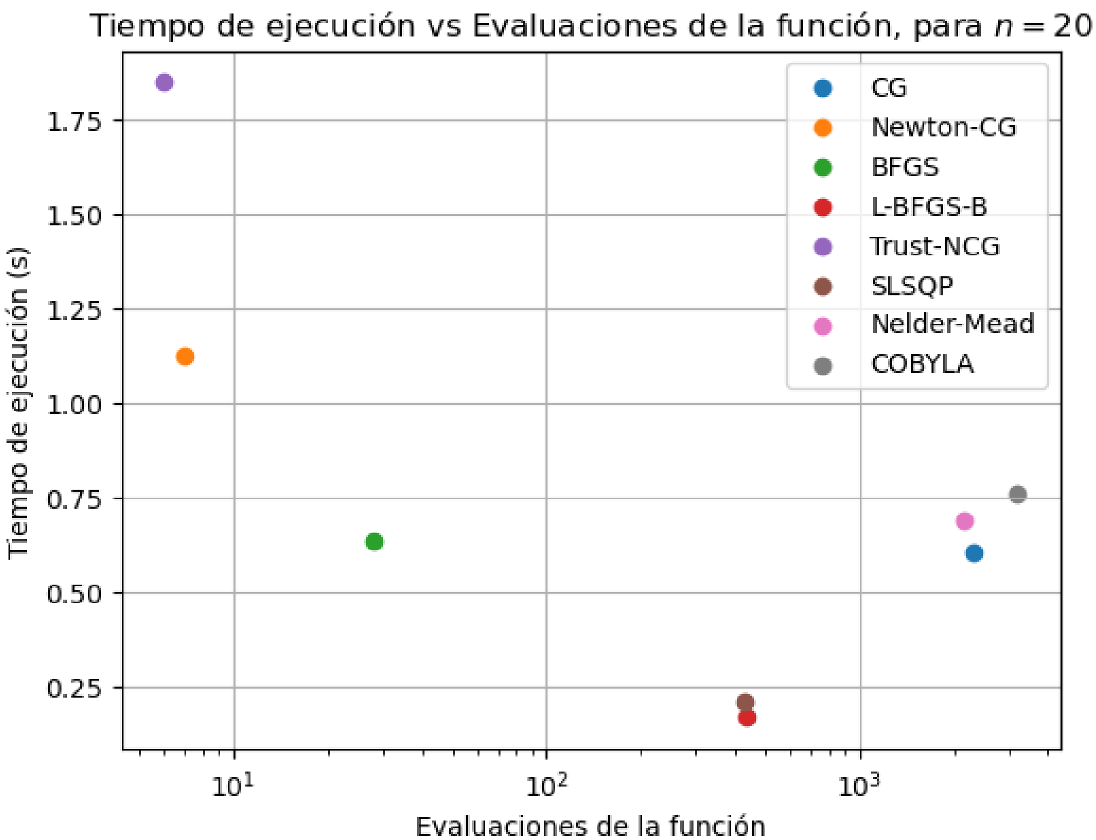
```python
[ ]: time_start = time.time()
```

```
sol_cobyla = minimize(area_func, F_init, method='COBYLA', bounds=bounds,␣
 ↪options={'maxiter': 100000})

time_end = time.time()
time_cobyla = np.append(time_cobyla, time_end - time_start)
Famp_cobyla = np.concatenate(([F0], sol_cobyla.x, [F1]))
error_cobyla = np.append(error_cobyla, np.linalg.norm(Famp_cobyla - Fcat))
```

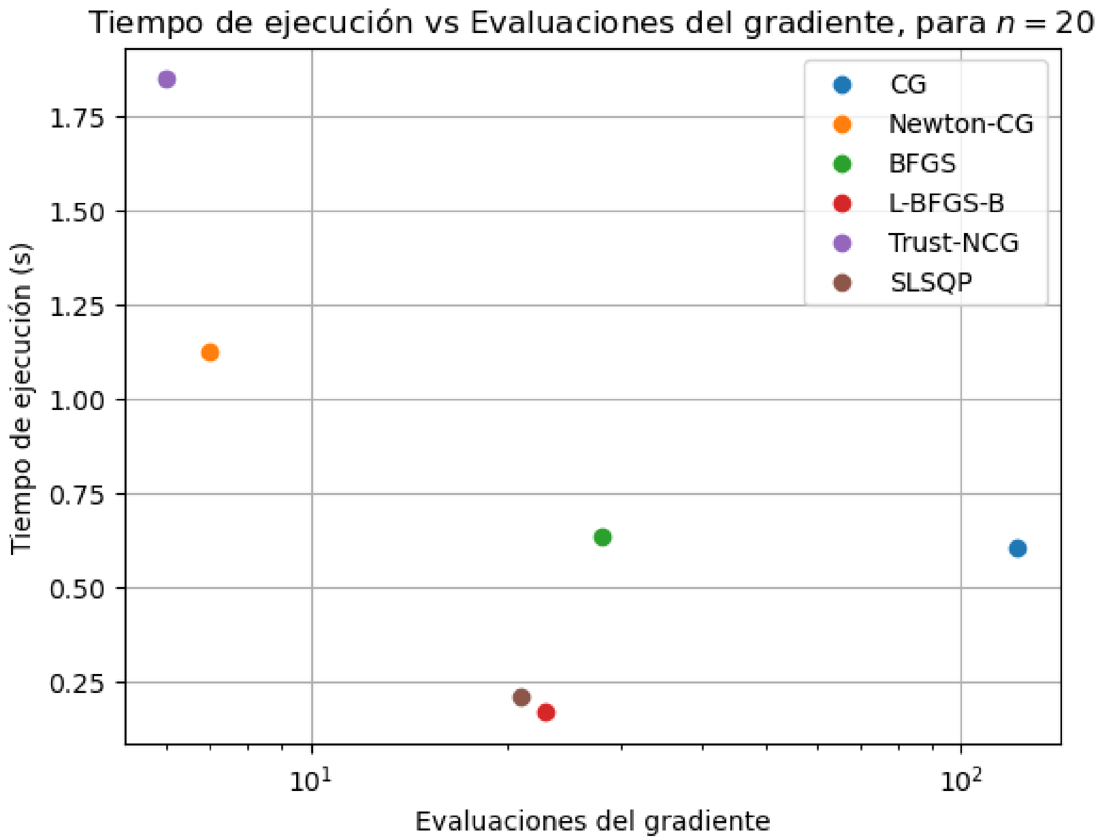
[ ]:
```
print(sol_cobyla.message)
print('Valor de la función:', sol_cobyla.fun)
# print('Resultado:', sol_cobyla.x)
print('Evaluaciones de la función:', sol_cobyla.nfev)
print('Tiempo de ejecución:', time_cobyla[-1])
print('El error para el método de COBYLA es:', error_cobyla[-1])
```

```
Optimization terminated successfully.
Valor de la función: 0.9537740652133743
Evaluaciones de la función: 3193
Tiempo de ejecución: 0.7569169998168945
El error para el método de COBYLA es: 0.013144272969841083
```

[ ]:
```
plt.scatter(optimizer_CG.nfev, time_CG[-1], label='CG')
plt.scatter(optimizer_Newton_CG.nfev, time_Newton_CG[-1], label='Newton-CG')
plt.scatter(optimizer_BFGS.nfev, time_BFGS[-1], label='BFGS')
plt.scatter(sol_LBFGSB.nfev, time_LBFGSB[-1], label='L-BFGS-B')
plt.scatter(sol_trustncg.nfev, time_trustncg[-1], label='Trust-NCG')
plt.scatter(sol_SLSQP.nfev, time_SLSQP[-1], label='SLSQP')
plt.scatter(sol_nelder.nfev, time_nelder[-1], label='Nelder-Mead')
plt.scatter(sol_cobyla.nfev, time_cobyla[-1], label='COBYLA')
plt.legend()
plt.ylabel('Tiempo de ejecución (s)')
plt.xlabel('Evaluaciones de la función')
plt.title('Tiempo de ejecución vs Evaluaciones de la función, para $n = {}$'.
 ↪format(n))
plt.xscale('log')
plt.grid(True)
plt.show()
```



[ ]:
```python
plt.scatter(optimizer_CG.njev, time_CG[-1], label='CG')
plt.scatter(optimizer_Newton_CG.njev, time_Newton_CG[-1], label='Newton-CG')
plt.scatter(optimizer_BFGS.njev, time_BFGS[-1], label='BFGS')
plt.scatter(sol_LBFGSB.njev, time_LBFGSB[-1], label='L-BFGS-B')
plt.scatter(sol_trustncg.njev, time_trustncg[-1], label='Trust-NCG')
plt.scatter(sol_SLSQP.njev, time_SLSQP[-1], label='SLSQP')
plt.legend()
plt.ylabel('Tiempo de ejecución (s)')
plt.xlabel('Evaluaciones del gradiente')
plt.title('Tiempo de ejecución vs Evaluaciones del gradiente, para $n = {}$'.
 ↪format(n))
plt.xscale('log')
plt.grid(True)
plt.show()
```



En este caso, siendo bajo el número de puntos y, por tanto, baja la dimensionalidad del problema, resulta barato computacionalmente evaluar la función un número elevado de veces, como se manifiesta de los resultados de métodos como, por ejemplo, COBYLA, Nelder-Mead o Gradiente Conjugado.

También es importante recalcar la gran diferencia que se encuentra entre los métodos de Newton (tanto Newton-Gradiente Conjugado como este mismo con la inclusión de las regiones de confianza) y le resto de métodos, sinebndo los basados en el método de Newton mucho más lentos, pese a evaluar tanto la función como el gradiente el menor número de veces. esto es debido a que son métodos que también requieren de evaluart la Hessiana, lo que aumenta significativamente el tiempo de computación necesario para resolver dichos problemas, especialmente en estos casos tan sencillos donde pueden no ser necesarios métodos tan sofisticados y es más fácil evaluar un número grande de veces la función.

Funciones con moderadas variables de salida: $n = 100$**.**

[ ]:
```
n = 100
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
```

```
bounds = np.vstack((lb, ub)).T
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)
```

1. **Método del *Steepest Descent* (descenso más pronunciado).**

[ ]:
```
# time_start = time.time()

# optimizer_steepest = SteepestDescent(area_func, F_init)
# optimizer_steepest.run()

# time_end = time.time()
# time_steepest = time_end - time_start
# Famp_steepest = np.concatenate(([F0], optimizer_steepest.x, [F1]))
```

[ ]:
```
# print(optimizer_steepest.message)
# print('Valor de la función:', optimizer_steepest.fun)
# # print('Resultado:', optimizer_steepest.x)
# print('Evaluaciones de la función:', optimizer_steepest.nfev)
# print('Tiempo de ejecución:', time_steepest)
```

2. ***Conjugate Gradient* (gradiente conjugado).**

[ ]:
```
# time_start = time.time()

# optimizer_CG = minimize(area_func, F_init, method='CG', options={'maxiter':␣
↪100000}) # no se pueden imponer restricciones

# time_end = time.time()
# time_CG = np.append(time_CG, time_end - time_start)
# Famp_CG = np.concatenate(([F0], optimizer_CG.x, [F1]))
# error_CG = np.append(error_CG, np.linalg.norm(Famp_CG - Fcat))
```

[ ]:
```
# print(optimizer_CG.message)
# print('Valor de la función:', optimizer_CG.fun)
# # print('Resultado:', optimizer_CG.x)
# print('Evaluaciones de la función:', optimizer_CG.nfev)
# print('Tiempo de ejecución:', time_CG[-1])
# print('Evaluaciones del gradiente:', optimizer_CG.njev)
# print('El error para el método de Newton-CG es:', error_CG[-1])
```

3. **Método de Newton de gradiente conjugado.**

[ ]:
```
# time_start = time.time()

# optimizer_Newton_CG = minimize(area_func, F_init, method='Newton-CG',␣
↪jac=area_func_grad, hess=area_func_hessian, options={'maxiter': 100000})

# time_end = time.time()
```

```
# time_Newton_CG = np.append(time_Newton_CG, time_end - time_start)
# Famp_Newton_CG = np.concatenate(([F0], optimizer_Newton_CG.x, [F1]))
# error_Newton_CG = np.append(error_Newton_CG, np.linalg.norm(Famp_Newton_CG -␣
↪Fcat))
```

[ ]:
```
# print(optimizer_Newton_CG.message)
# print('Valor de la función:', optimizer_Newton_CG.fun)
# # print('Resultado:', optimizer_Newton_CG.x)
# print('Evaluaciones de la función:', optimizer_Newton_CG.nfev)
# print('Tiempo de ejecución:', time_Newton_CG[-1])
# print('Evaluaciones del gradiente:', optimizer_Newton_CG.njev)
# print('El error para el método de Newton-CG es:', error_Newton_CG[-1])
```

4. **Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS)** (quasi-Newton).

[ ]:
```
# time_start = time.time()

# optimizer_BFGS = minimize(area_func, F_init, method='BFGS',␣
↪jac=area_func_grad, options={'maxiter': 100000})

# time_end = time.time()
# time_BFGS = np.append(time_BFGS, time_end - time_start)
# Famp_BFGS = np.concatenate(([F0], optimizer_BFGS.x, [F1]))
# error_BFGS = np.append(error_BFGS, np.linalg.norm(Famp_BFGS - Fcat))
```

[ ]:
```
# print(optimizer_BFGS.message)
# print('Valor de la función:', optimizer_BFGS.fun)
# # print('Resultado:', optimizer_BFGS.x)
# print('Evaluaciones de la función:', optimizer_BFGS.nfev)
# print('Tiempo de ejecución:', time_BFGS[-1])
# print('Evaluaciones del gradiente:', optimizer_BFGS.njev)
# print('El error para el método de BFGS es:', error_BFGS[-1])
```

5. ***Limited-memory BFGS with Bound constraints* (L-BFGS-B)**.

[ ]:
```
# time_start = time.time()

# sol_LBFGSB = minimize(area_func, F_init, method='L-BFGS-B', bounds=bounds,␣
↪options={'maxiter': 100000})

# time_end = time.time()
# time_LBFGSB = np.append(time_LBFGSB, time_end - time_start)
# Famp_LBFGSB = np.concatenate(([F0], sol_LBFGSB.x, [F1]))
# error_LBFGSB = np.append(error_LBFGSB, np.linalg.norm(Famp_LBFGSB - Fcat))
```

[ ]:
```
# print(sol_LBFGSB.message)
# print('Valor de la función:', sol_LBFGSB.fun)
# # print('Resultado:', sol_LBFGSB.x)
```

```
# print('Evaluaciones de la función:', sol_LBFGSB.nfev)
# print('Tiempo de ejecución:', time_LBFGSB[-1])
# print('Evaluaciones del gradiente:', sol_LBFGSB.njev)
# print('El error para el método de L-BFGS-B es:', error_LBFGSB[-1])
```

6. ***Newton Conjugate Gradient Trust-Region***.

```
[ ]: # time_start = time.time()

# sol_trustncg = minimize(area_func, F_init, method='trust-ncg',␣
↪jac=area_func_grad, hess=area_func_hessian, options={'maxiter': 100000})

# time_end = time.time()
# time_trustncg = np.append(time_trustncg, time_end - time_start)
# Famp_trustncg = np.concatenate(([F0], sol_trustncg.x, [F1]))
# error_trustncg = np.append(error_trustncg, np.linalg.norm(Famp_trustncg -␣
↪Fcat))
```

```
[ ]: # print(sol_trustncg.message)
# print('Valor de la función:', sol_trustncg.fun)
# # print('Resultado:', sol_trustncg.x)
# print('Evaluaciones de la función:', sol_trustncg.nfev)
# print('Tiempo de ejecución:', time_trustncg[-1])
# print('Evaluaciones del gradiente:', sol_trustncg.njev)
# print('El error para el método de trust-ncg es:', error_trustncg[-1])
```

7. ***Sequential Least Squares Programming*** **(SLSQP)**.

```
[ ]: # start = time.time()

# sol_SLSQP = minimize(area_func, F_init, method='SLSQP', bounds=bounds,␣
↪options={'maxiter': 100000})

# end = time.time()
# time_SLSQP = np.append(time_SLSQP, end - start)
# Famp_SLSQP = np.concatenate(([F0], sol_SLSQP.x, [F1]))
# error_SLSQP = np.append(error_SLSQP, np.linalg.norm(Famp_SLSQP - Fcat))
```

```
[ ]: # print(sol_SLSQP.message)
# print('Valor de la función:', sol_SLSQP.fun)
# # print('Resultado:', sol_SLSQP.x)
# print('Evaluaciones de la función:', sol_SLSQP.nfev)
# print('Tiempo de ejecución:', time_SLSQP[-1])
# print('Evaluaciones del gradiente:', sol_SLSQP.njev)
# print('El error para el método de SLSQP es:', error_SLSQP[-1])
```

8. **Nelder-Mead**.

```
[ ]: # time_start = time.time()

     # sol_nelder = minimize(area_func, F_init, method='Nelder-Mead', bounds=bounds,␣
     ↪options={'maxiter': 1000000})

     # time_end = time.time()
     # time_nelder = np.append(time_nelder, time_end - time_start)
     # Famp_nelder = np.concatenate(([F0], sol_nelder.x, [F1]))
     # error_nelder = np.append(error_nelder, np.linalg.norm(Famp_nelder - Fcat))
```

```
[ ]: # print(sol_nelder.message)
     # print('Valor de la función:', sol_nelder.fun)
     # # print('Resultado:', sol_nelder.x)
     # print('Evaluaciones de la función:', sol_nelder.nfev)
     # print('Tiempo de ejecución:', time_nelder[-1])
     # print('El error para el método de Nelder-Mead es:', error_nelder[-1])
```

9. ***Constrained Optimization BY Linear Approximation*** **(COBYLA)**.

```
[ ]: # time_start = time.time()

     # sol_cobyla = minimize(area_func, F_init, method='COBYLA', bounds=bounds,␣
     ↪options={'maxiter': 1000000})

     # time_end = time.time()
     # time_cobyla = np.append(time_cobyla, time_end - time_start)
     # Famp_cobyla = np.concatenate(([F0], sol_cobyla.x, [F1]))
     # error_cobyla = np.append(error_cobyla, np.linalg.norm(Famp_cobyla - Fcat))
```

```
[ ]: # print(sol_cobyla.message)
     # print('Valor de la función:', sol_cobyla.fun)
     # # print('Resultado:', sol_cobyla.x)
     # print('Evaluaciones de la función:', sol_cobyla.nfev)
     # print('Tiempo de ejecución:', time_cobyla[-1])
     # print('El error para el método de COBYLA es:', error_cobyla[-1])
```

```
[ ]: # plt.scatter(optimizer_CG.nfev, time_CG[-1], label='CG')
     # plt.scatter(optimizer_Newton_CG.nfev, time_Newton_CG[-1], label='Newton-CG')
     # plt.scatter(optimizer_BFGS.nfev, time_BFGS[-1], label='BFGS')
     # plt.scatter(sol_LBFGSB.nfev, time_LBFGSB[-1], label='L-BFGS-B')
     # plt.scatter(sol_trustncg.nfev, time_trustncg[-1], label='Trust-NCG')
     # plt.scatter(sol_SLSQP.nfev, time_SLSQP[-1], label='SLSQP')
     # plt.scatter(sol_nelder.nfev, time_nelder[-1], label='Nelder-Mead')
     # plt.scatter(sol_cobyla.nfev, time_cobyla[-1], label='COBYLA')
     # plt.legend()
     # plt.ylabel('Tiempo de ejecución (s)')
     # plt.xlabel('Evaluaciones de la función')
     # plt.xscale('log')
```

```
# plt.title('Tiempo de ejecución vs Evaluaciones de la función, para $n = {}$'.
↪format(n))
# plt.grid(True)
# plt.show()
```

[ ]:
```
# plt.scatter(optimizer_CG.njev, time_CG[-1], label='CG')
# plt.scatter(optimizer_Newton_CG.njev, time_Newton_CG[-1], label='Newton-CG')
# plt.scatter(optimizer_BFGS.njev, time_BFGS[-1], label='BFGS')
# plt.scatter(sol_LBFGSB.njev, time_LBFGSB[-1], label='L-BFGS-B')
# plt.scatter(sol_trustncg.njev, time_trustncg[-1], label='Trust-NCG')
# plt.scatter(sol_SLSQP.njev, time_SLSQP[-1], label='SLSQP')
# plt.legend()
# plt.ylabel('Tiempo de ejecución (s)')
# plt.xlabel('Evaluaciones del gradiente')
# plt.title('Tiempo de ejecución vs Evaluaciones del gradiente, para $n = {}$'.
↪format(n))
# plt.xscale('log')
# plt.grid(True)
# plt.show()
```

Una vez se aumenta considerablemente el número de puntos, cambia la tendencia de la solución en función del método empleado. Se aprecia que métodos que no hacen uso del gradiente y evaluan la función un número grande de ves, como son el COBYLA o el Nelder-Mead suponen un coste elevadísimo que incrementa el tiempo necesario para alcanzar la solución del problema.

Por otro lado, se aprecia como métodos más complejos pero que pueden resultar más robustos (BGFS, L-BFGS-B o SLSQP), mantienen unos tiempos de computación excelentes, pese a haber aumentado en gran medida la complicación del problema.

Funciones muchas variables de salida: $n = 400$.

[ ]:
```
n = 400
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)
```

1. **Método del *Steepest Descent* (descenso más pronunciado).**

[ ]:
```
# time_start = time.time()

# optimizer_steepest = SteepestDescent(area_func, F_init)
# optimizer_steepest.run()

# time_end = time.time()
```

```
# time_steepest = time_end - time_start
# Famp_steepest = np.concatenate(([F0], optimizer_steepest.x, [F1]))
```

[ ]:
```
# print(optimizer_steepest.message)
# print('Valor de la función:', optimizer_steepest.fun)
# # print('Resultado:', optimizer_steepest.x)
# print('Evaluaciones de la función:', optimizer_steepest.nfev)
# print('Tiempo de ejecución:', time_steepest)
```

2. ***Conjugate Gradient*** **(gradiente conjugado)**.

[ ]:
```
# time_start = time.time()

# optimizer_CG = minimize(area_func, F_init, method='CG', options={'maxiter':␣
↪100000}) # no se pueden imponer restricciones

# time_end = time.time()
# time_CG = np.append(time_CG, time_end - time_start)
# Famp_CG = np.concatenate(([F0], optimizer_CG.x, [F1]))
# error_CG = np.append(error_CG, np.linalg.norm(Famp_CG - Fcat))
```

[ ]:
```
# print(optimizer_CG.message)
# print('Valor de la función:', optimizer_CG.fun)
# # print('Resultado:', optimizer_CG.x)
# print('Evaluaciones de la función:', optimizer_CG.nfev)
# print('Tiempo de ejecución:', time_CG[-1])
# print('Evaluaciones del gradiente:', optimizer_CG.njev)
# print('El error para el método de Newton-CG es:', error_CG[-1])
```

3. **Método de Newton de gradiente conjugado**.

[ ]:
```
# time_start = time.time()

# optimizer_Newton_CG = minimize(area_func, F_init, method='Newton-CG',␣
↪jac=area_func_grad, hess=area_func_hessian, options={'maxiter': 100000})

# time_end = time.time()
# time_Newton_CG = np.append(time_Newton_CG, time_end - time_start)
# Famp_Newton_CG = np.concatenate(([F0], optimizer_Newton_CG.x, [F1]))
# error_Newton_CG = np.append(error_Newton_CG, np.linalg.norm(Famp_Newton_CG -␣
↪Fcat))
```

[ ]:
```
# print(optimizer_Newton_CG.message)
# print('Valor de la función:', optimizer_Newton_CG.fun)
# # print('Resultado:', optimizer_Newton_CG.x)
# print('Evaluaciones de la función:', optimizer_Newton_CG.nfev)
# print('Tiempo de ejecución:', time_Newton_CG[-1])
# print('Evaluaciones del gradiente:', optimizer_Newton_CG.njev)
```

```
# print('El error para el método de Newton-CG es:', error_Newton_CG[-1])
```

4. **Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS)** (quasi-Newton).

```
[ ]: # time_start = time.time()

# optimizer_BFGS = minimize(area_func, F_init, method='BFGS',␣
↪jac=area_func_grad, options={'maxiter': 100000})

# time_end = time.time()
# time_BFGS = np.append(time_BFGS, time_end - time_start)
# Famp_BFGS = np.concatenate(([F0], optimizer_BFGS.x, [F1]))
# error_BFGS = np.append(error_BFGS, np.linalg.norm(Famp_BFGS - Fcat))
```

```
[ ]: # print(optimizer_BFGS.message)
# print('Valor de la función:', optimizer_BFGS.fun)
# # print('Resultado:', optimizer_BFGS.x)
# print('Evaluaciones de la función:', optimizer_BFGS.nfev)
# print('Tiempo de ejecución:', time_BFGS[-1])
# print('Evaluaciones del gradiente:', optimizer_BFGS.njev)
# print('El error para el método de BFGS es:', error_BFGS[-1])
```

5. ***Limited-memory BFGS with Bound constraints*** **(L-BFGS-B)**.

```
[ ]: # time_start = time.time()

# sol_LBFGSB = minimize(area_func, F_init, method='L-BFGS-B', bounds=bounds,␣
↪options={'maxiter': 100000})

# time_end = time.time()
# time_LBFGSB = np.append(time_LBFGSB, time_end - time_start)
# Famp_LBFGSB = np.concatenate(([F0], sol_LBFGSB.x, [F1]))
# error_LBFGSB = np.append(error_LBFGSB, np.linalg.norm(Famp_LBFGSB - Fcat))
```

```
[ ]: # print(sol_LBFGSB.message)
# print('Valor de la función:', sol_LBFGSB.fun)
# # print('Resultado:', sol_LBFGSB.x)
# print('Evaluaciones de la función:', sol_LBFGSB.nfev)
# print('Tiempo de ejecución:', time_LBFGSB[-1])
# print('Evaluaciones del gradiente:', sol_LBFGSB.njev)
# print('El error para el método de L-BFGS-B es:', error_LBFGSB[-1])
```

6. ***Newton Conjugate Gradient Trust-Region***.

```
[ ]: # time_start = time.time()

# sol_trustncg = minimize(area_func, F_init, method='trust-ncg',␣
↪jac=area_func_grad, hess=area_func_hessian, options={'maxiter': 100000})
```

```
# time_end = time.time()
# time_trustncg = np.append(time_trustncg, time_end - time_start)
# Famp_trustncg = np.concatenate(([F0], sol_trustncg.x, [F1]))
# error_trustncg = np.append(error_trustncg, np.linalg.norm(Famp_trustncg -␣
↪Fcat))
```

[ ]:
```
# print(sol_trustncg.message)
# print('Valor de la función:', sol_trustncg.fun)
# # print('Resultado:', sol_trustncg.x)
# print('Evaluaciones de la función:', sol_trustncg.nfev)
# print('Tiempo de ejecución:', time_trustncg[-1])
# print('Evaluaciones del gradiente:', sol_trustncg.njev)
# print('El error para el método de trust-ncg es:', error_trustncg[-1])
```

7. ***Sequential Least Squares Programming*** **(SLSQP)**.

[ ]:
```
# start = time.time()

# sol_SLSQP = minimize(area_func, F_init, method='SLSQP', bounds=bounds,␣
↪options={'maxiter': 100000})

# end = time.time()
# time_SLSQP = np.append(time_SLSQP, end - start)
# Famp_SLSQP = np.concatenate(([F0], sol_SLSQP.x, [F1]))
# error_SLSQP = np.append(error_SLSQP, np.linalg.norm(Famp_SLSQP - Fcat))
```

[ ]:
```
# print(sol_SLSQP.message)
# print('Valor de la función:', sol_SLSQP.fun)
# # print('Resultado:', sol_SLSQP.x)
# print('Evaluaciones de la función:', sol_SLSQP.nfev)
# print('Tiempo de ejecución:', time_SLSQP[-1])
# print('Evaluaciones del gradiente:', sol_SLSQP.njev)
# print('El error para el método de SLSQP es:', error_SLSQP[-1])
```

8. **Nelder-Mead**.

[ ]:
```
# time_start = time.time()

# sol_nelder = minimize(area_func, F_init, method='Nelder-Mead', bounds=bounds,␣
↪options={'maxiter': 100000})

# time_end = time.time()
# time_nelder = np.append(time_nelder, time_end - time_start)
# Famp_nelder = np.concatenate(([F0], sol_nelder.x, [F1]))
# error_nelder = np.append(error_nelder, np.linalg.norm(Famp_nelder - Fcat))
```

[ ]:
```
# print(sol_nelder.message)
# print('Valor de la función:', sol_nelder.fun)
# # print('Resultado:', sol_nelder.x)
# print('Evaluaciones de la función:', sol_nelder.nfev)
# print('Tiempo de ejecución:', time_nelder[-1])
# print('El error para el método de Nelder-Mead es:', error_nelder[-1])
```

9. ***Constrained Optimization BY Linear Approximation*** **(COBYLA)**.

[ ]:
```
# time_start = time.time()

# sol_cobyla = minimize(area_func, F_init, method='COBYLA', bounds=bounds,␣
↪options={'maxiter': 100000})

# time_end = time.time()
# time_cobyla = np.append(time_cobyla, time_end - time_start)
# Famp_cobyla = np.concatenate(([F0], sol_cobyla.x, [F1]))
# error_cobyla = np.append(error_cobyla, np.linalg.norm(Famp_cobyla - Fcat))
```

[ ]:
```
# print(sol_cobyla.message)
# print('Valor de la función:', sol_cobyla.fun)
# # print('Resultado:', sol_cobyla.x)
# print('Evaluaciones de la función:', sol_cobyla.nfev)
# print('Tiempo de ejecución:', time_cobyla[-1])
# print('El error para el método de COBYLA es:', error_cobyla[-1])
```

[ ]:
```
# plt.scatter(optimizer_CG.nfev, time_CG[-1], label='CG')
# plt.scatter(optimizer_Newton_CG.nfev, time_Newton_CG[-1], label='Newton-CG')
# plt.scatter(optimizer_BFGS.nfev, time_BFGS[-1], label='BFGS')
# plt.scatter(sol_LBFGSB.nfev, time_LBFGSB[-1], label='L-BFGS-B')
# plt.scatter(sol_trustncg.nfev, time_trustncg[-1], label='Trust-NCG')
# plt.scatter(sol_SLSQP.nfev, time_SLSQP[-1], label='SLSQP')
# plt.scatter(sol_nelder.nfev, time_nelder[-1], label='Nelder-Mead')
# plt.scatter(sol_cobyla.nfev, time_cobyla[-1], label='COBYLA')
# plt.legend()
# plt.ylabel('Tiempo de ejecución (s)')
# plt.xlabel('Evaluaciones de la función')
# plt.title('Tiempo de ejecución vs Evaluaciones de la función, para $n = {}$'.
↪format(n))
# plt.xscale('log')
# plt.grid(True)
# plt.show()
```

[ ]:
```
# plt.scatter(optimizer_CG.njev, time_CG[-1], label='CG')
# plt.scatter(optimizer_Newton_CG.njev, time_Newton_CG[-1], label='Newton-CG')
# plt.scatter(optimizer_BFGS.njev, time_BFGS[-1], label='BFGS')
# plt.scatter(sol_LBFGSB.njev, time_LBFGSB[-1], label='L-BFGS-B')
# plt.scatter(sol_trustncg.njev, time_trustncg[-1], label='Trust-NCG')
```

```
# plt.scatter(sol_SLSQP.njev, time_SLSQP[-1], label='SLSQP')
# plt.legend()
# plt.ylabel('Tiempo de ejecución (s)')
# plt.xlabel('Evaluaciones del gradiente')
# plt.title('Tiempo de ejecución vs Evaluaciones del gradiente, para $n = {}$'.
 ↪format(n))
# plt.xscale('log')
# plt.grid(True)
# plt.show()
```

Para este último caso, se han aumentado los puntos muy por encima de lo que sería incluso recomendable para alcanzar la solución, solo a fin de estudiar el comportamiento de los algoritmos cuando se les fuerza en una función de alta dimensionalidad.

Por encima de todos los resultados, se distingue el mal comportamiento del algoritmo del Gradiente Conjugado. Esto era de esperar ya que, siendo un método tan simple, al aumentar la complicación del problema responde peor de lo que lo hacen el resto, siendo el que requiere más evaluaciones de la función, más del gradiente y a su vez el que tarda más tiempo en converger a una solución.

Por otro lado, vuelven a destacar los métodos BGFS, L-BFGS-B y SLSQP por su rapidez, manteniendo la robustez que antes se comentaba.

[ ]:
```
# plt.plot(time_CG, error_CG, label = 'CG')
# plt.plot(time_Newton_CG, error_Newton_CG, label = 'Newton-CG')
# plt.plot(time_BFGS, error_BFGS, label = 'BFGS')
# plt.plot(time_LBFGSB, error_LBFGSB, label = 'L-BFGS-B')
# plt.plot(time_trustncg, error_trustncg, label = 'Trust-NCG')
# plt.plot(time_SLSQP, error_SLSQP, label = 'SLSQP')
# plt.plot(time_nelder, error_nelder, label = 'Nelder-Mead')
# # plt.plot(time_cobyla, error_cobyla, label = 'COBYLA')
# plt.legend()
# plt.yscale('log')
# plt.xscale('log')
# plt.ylabel('Error')
# plt.xlabel('tiempo (s)')
# plt.title('Error en función de n')
# plt.grid(True)
# plt.show()
```

Influencia del valor de arranque para la iteración. Otro parámetro que puede influir en las características de la solución obtenida es el valor que se escoja de arranque en la iteración, lo que equivale al punto escogido desde el que arrancar el método de basado en gradiente que se escoja.

En principio, se ha escogido para este valor un caso de $F(z) = F_0 \quad, \forall z \in [0, 1]$, pero a continuación se propone inicializar en distintos valores, que se detallan a continuación.

En todos estos se considera $F_0 = 1$, por lo que se omite en las expresiones.

[58]:
```
n = 20
F_0 = 1
F0 = F_0
F1 = F_0

z = np.linspace(0, 1, n)

lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)
```

1. $F(z) = 1 \quad , \forall z \in [0, 1]$

[59]:
```
F_init = np.empty(n-2)
F_init.fill(F_0)

sol_F0 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_F0 = np.concatenate(([F0], sol_F0.x, [F1]))
error_F0 = np.linalg.norm(Famp_F0 - Fcat)/np.linalg.norm(Fcat)
```

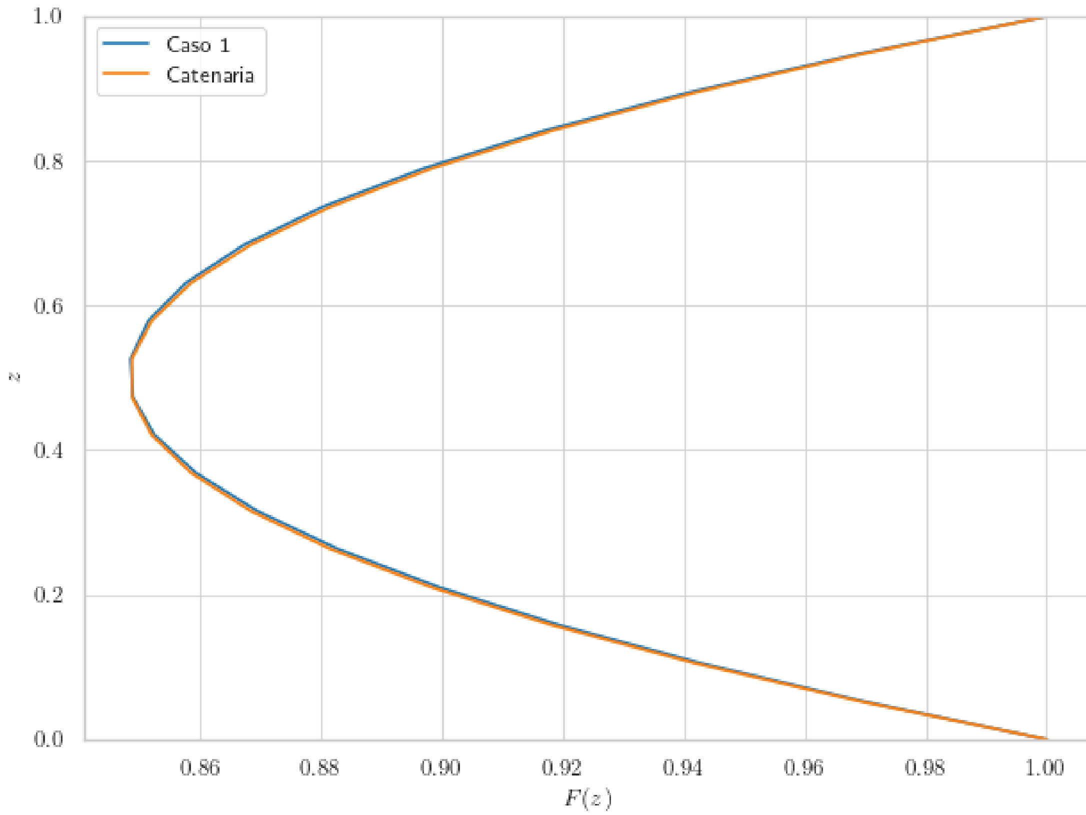
[60]:
```
print(sol_F0.message)
# print('Valor de la función:', sol_F0.fun)
# print('Resultado:', sol_F0.x)
print('Evaluaciones de la función:', sol_F0.nfev)
print('El error para F_inic=1 es:', error_F0*100)
```

```
Optimization terminated successfully
Evaluaciones de la función: 429
El error para F_inic=1 es: 0.10218191336893907
```

Como se ha comprobado anteriormente, al alcanzar un valor del error de $≈ $4 \cdot 10@\{-3\}\$ seconsideraquelafuncinhallegadoalmnimoquepermitenlascondicionesdelproblema.Portanto,paracomp$

[61]:
```
plt.figure()
plt.plot(Famp_F0, z, label= 'Caso 1')
# plt.plot(Famp_0, z, label= 'Caso 2')
# plt.plot(Famp_F06, z, label= 'Caso 3')
# plt.plot(Famp_F2, z, label= 'Caso 4')
# plt.plot(Famp_F10, z, label= 'Caso 5')
# plt.plot(Famp_rand, z, label= 'Caso 6')
plt.plot(Fcat, z, label='Catenaria')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()
```

```
# plt.savefig('Figuras/sol_cambioF.pdf', format='pdf')
plt.show()
```



2. $F(z) = 0 \quad , \forall z \in [0, 1]$

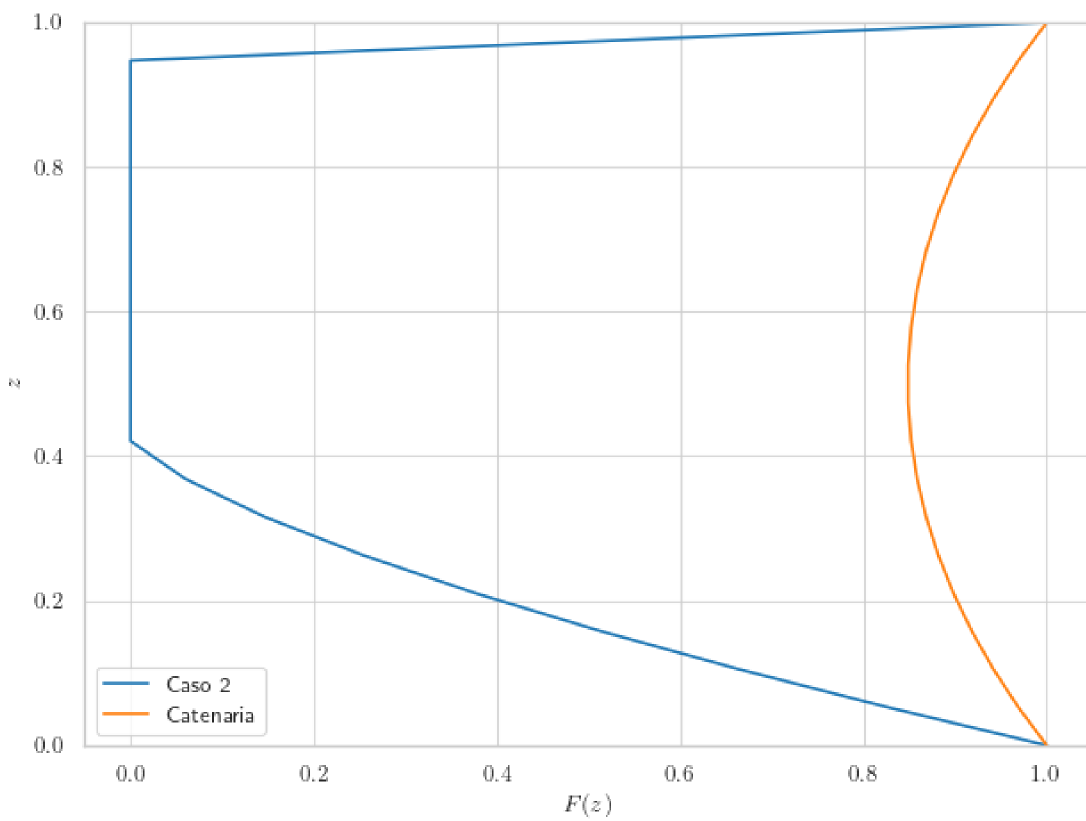
[62]:
```
F_init = np.zeros(n-2)

sol_0 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_0 = np.concatenate(([F0], sol_0.x, [F1]))
error_0 = np.linalg.norm(Famp_0 - Fcat)/np.linalg.norm(Fcat)
```

[63]:
```
print(sol_0.message)
# print('Valor de la función:', sol_0.fun)
# print('Resultado:', sol_0.x)
print('Evaluaciones de la función:', sol_0.nfev)
print('El error para F_inic=0 es:', error_0*100)
```

```
Optimization terminated successfully
Evaluaciones de la función: 249
El error para F_inic=0 es: 80.90899638820909
```

Como se ha mencionado antes, aunque el resultado de la iteración indique que se ha conseguido minimizar la función, del valor del error se observa que no es así. Esto se debe a que los valores de arranque pueden condicionar enormemente la capacidad de llegar a una solución válida.

[64]:
```python
plt.figure()
# plt.plot(Famp_F0, z, label= 'Caso 1')
plt.plot(Famp_0, z, label= 'Caso 2')
# plt.plot(Famp_F06, z, label= 'Caso 3')
# plt.plot(Famp_F2, z, label= 'Caso 4')
# plt.plot(Famp_F10, z, label= 'Caso 5')
# plt.plot(Famp_rand, z, label= 'Caso 6')
plt.plot(Fcat, z, label='Catenaria')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()
# plt.savefig('Figuras/sol_cambioF.pdf', format='pdf')
plt.show()
```



3. $F(z) \leq 0.6 \quad , \forall z \in [0, 1]$

[65]:

```
F_init = np.empty(n-2)
F_init.fill(F_0 * 0.6)

sol_F06 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_F06 = np.concatenate(([F0], sol_F06.x, [F1]))
error_F06 = np.linalg.norm(Famp_F06 - Fcat)/np.linalg.norm(Fcat)
```

[66]:

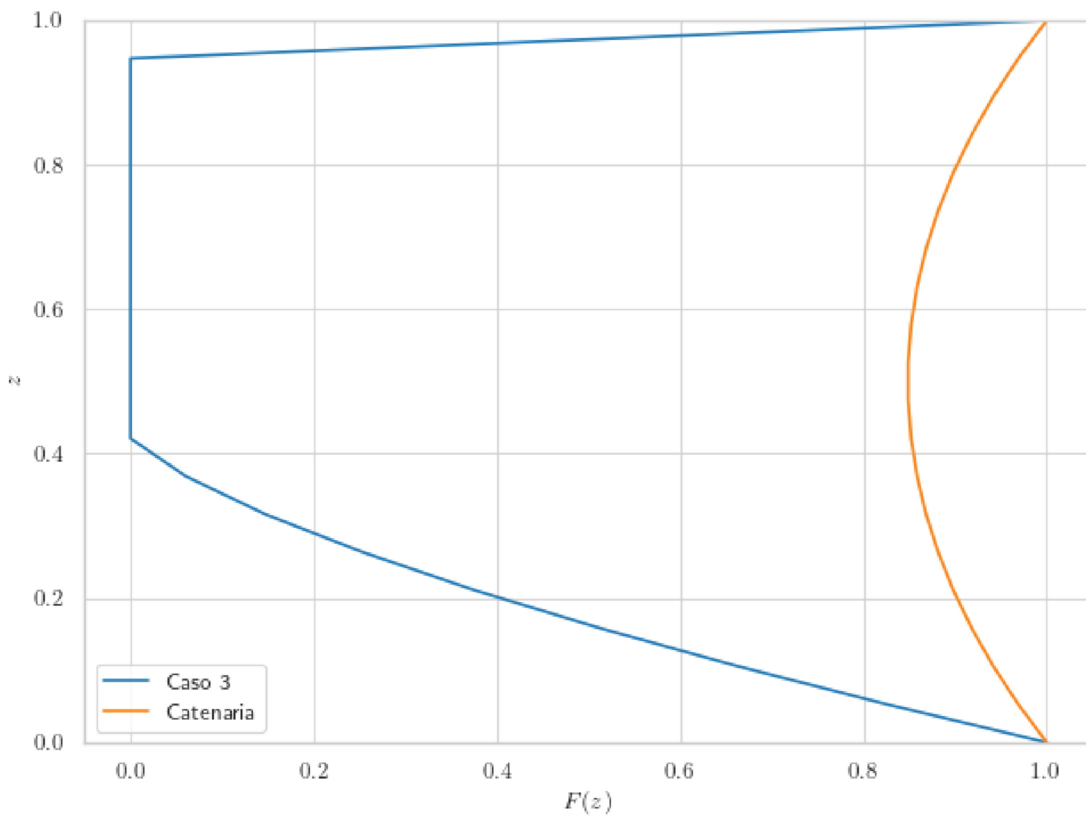
```
print(sol_F06.message)
# print('Valor de la función:', sol_F06.fun)
# print('Resultado:', sol_F06.x)
print('Evaluaciones de la función:', sol_F06.nfev)
print('El error para F_inic=0.6 es:', error_F06*100)
# print('Valor del area:', area_func(sol_F06.x))
```

```
Optimization terminated successfully
Evaluaciones de la función: 739
El error para F_inic=0.6 es: 80.89944306194334
```

Al igual que en el caso anterior, se encuentra que para valores iniciales de la iteración demasiado cercanos al origen, el algoritmo no es capaz de converger a la solución correcta.

[67]:

```
plt.figure()
# plt.plot(Famp_F0, z, label= 'Caso 1')
# plt.plot(Famp_0, z, label= 'Caso 2')
plt.plot(Famp_F06, z, label= 'Caso 3')
# plt.plot(Famp_F2, z, label= 'Caso 4')
# plt.plot(Famp_F10, z, label= 'Caso 5')
# plt.plot(Famp_rand, z, label= 'Caso 6')
plt.plot(Fcat, z, label='Catenaria')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()
plt.savefig('Figuras/sol_cambioF0.pdf', format='pdf')
plt.show()
```



4. $0.6 < F(z) < 2 \quad ,\forall z \in [0,1]$

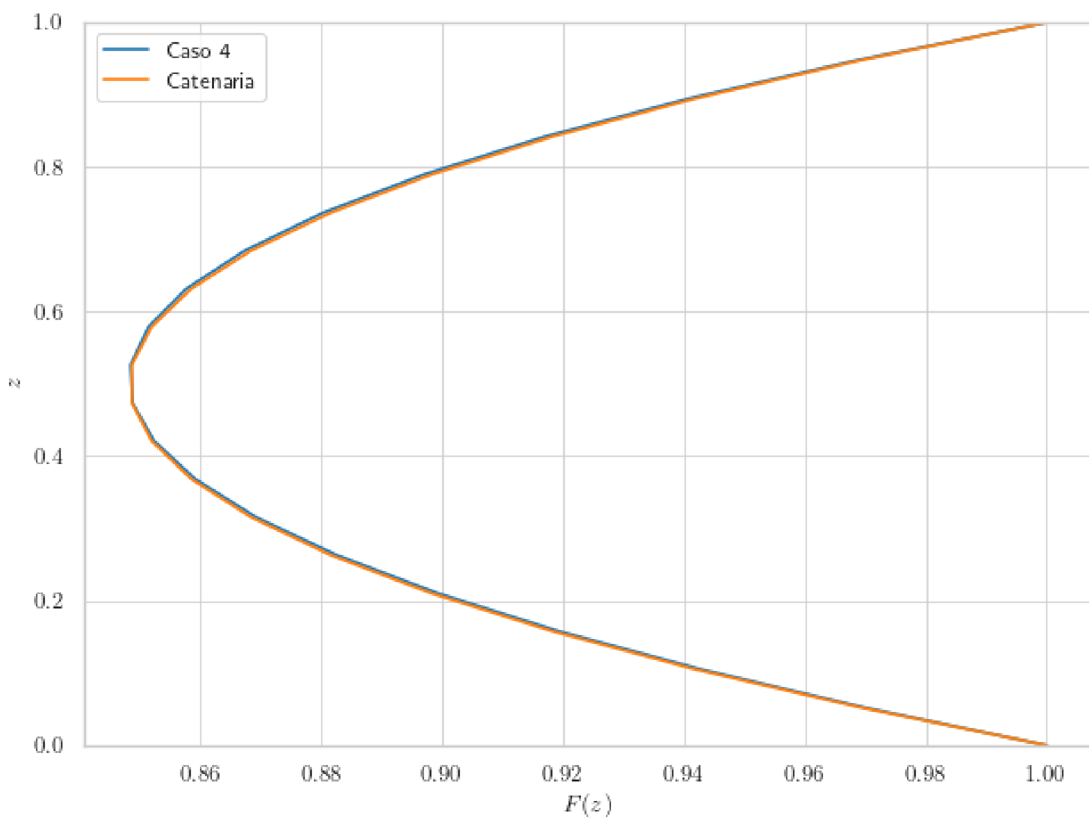
[68]:
```
F_init = np.empty(n-2)
F_init.fill(F_0 * 2)

sol_F2 = minimize(area_func, F_init, bounds=bounds, method='SLSQP',␣
 ↪options={'maxiter': 100000})
Famp_F2 = np.concatenate(([F0], sol_F2.x, [F1]))
error_F2 = np.linalg.norm(Famp_F2 - Fcat)/np.linalg.norm(Fcat)
```

[69]:
```
print(sol_F2.message)
# print('Valor de la función:', sol_F2.fun)
# print('Resultado:', sol_F2.x)
print('Evaluaciones de la función:', sol_F2.nfev)
print('El error para F_inic=2 es:', error_F2*100)
```

```
Optimization terminated successfully
Evaluaciones de la función: 615
El error para F_inic=2 es: 0.09260034305913725
```

Como podía llegar a inferirse, no es necesario que el algoritmo arranque exactamente en el valor de F_0, ya que eso provocaría muy poca robustez en la resolución del problema. Existe un intervalo que llamaremos “zona óptima de arranque” en la que los valores de la función pueden inicializarse y el algoritmo conseguirá converger para llegar a la solución adecuada.

[70]:
```
plt.figure()
# plt.plot(Famp_F0, z, label= 'Caso 1')
# plt.plot(Famp_0, z, label= 'Caso 2')
# plt.plot(Famp_F06, z, label= 'Caso 3')
plt.plot(Famp_F2, z, label= 'Caso 4')
# plt.plot(Famp_F10, z, label= 'Caso 5')
# plt.plot(Famp_rand, z, label= 'Caso 6')
plt.plot(Fcat, z, label='Catenaria')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()
# plt.savefig('Figuras/sol_cambioF.pdf', format='pdf')
plt.show()
```



5. $F(z) \gg 2 \quad , \forall z \in [0, 1]$

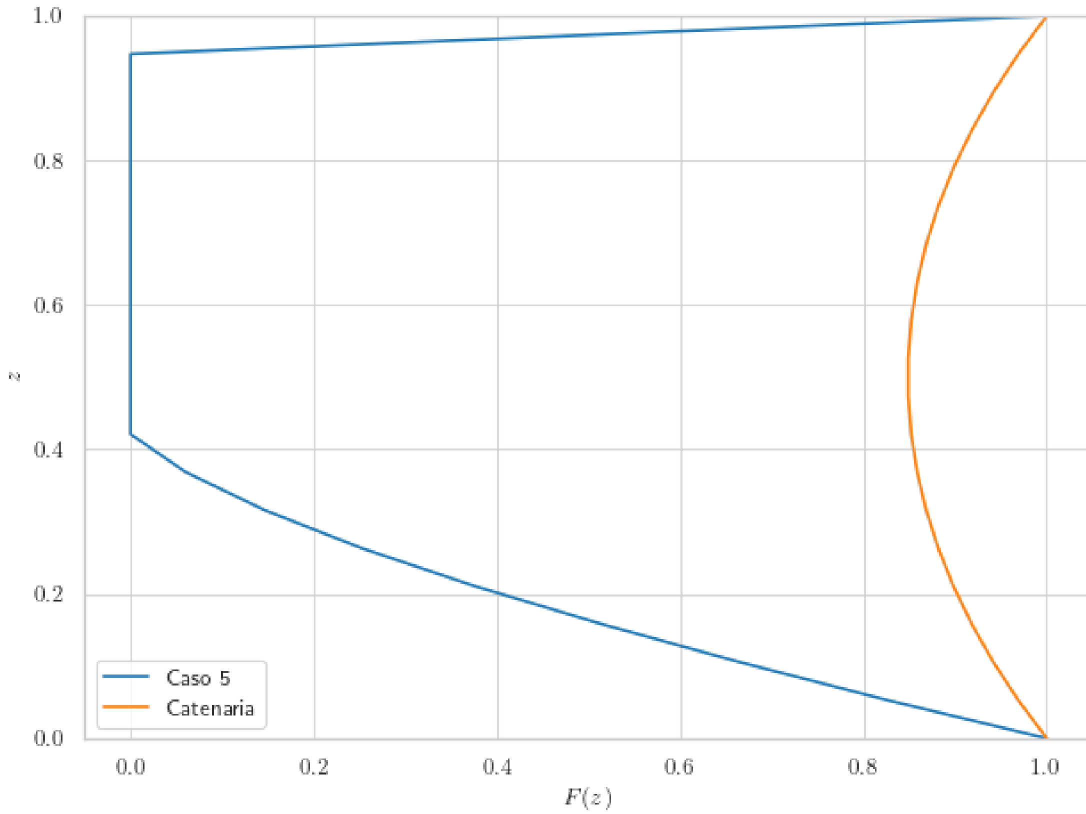
[71]:
```
F_init = np.empty(n-2)
F_init.fill(F_0 * 10)

sol_F10 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_F10 = np.concatenate(([F0], sol_F10.x, [F1]))
error_F10 = np.linalg.norm(Famp_F10 - Fcat)/np.linalg.norm(Fcat)
```

[72]:
```
print(sol_F10.message)
# print('Valor de la función:', sol_F10.fun)
# print('Resultado:', sol_F10.x)
print('Evaluaciones de la función:', sol_F10.nfev)
print('El error para F_inic=10 es:', error_F10*100)
```

```
Optimization terminated successfully
Evaluaciones de la función: 1013
El error para F_inic=10 es: 80.9021047991489
```

Al igual que en los puntos por debajo de la cota inferior de la zona óptima de arranque, los que se sitúan por encima llegan a una solución inadecuada, y esto se mantiene para todos los valores mayores que 2, según el análisis que se ha llevado a cabo.

[73]:
```
plt.figure()
# plt.plot(Famp_F0, z, label= 'Caso 1')
# plt.plot(Famp_0, z, label= 'Caso 2')
# plt.plot(Famp_F06, z, label= 'Caso 3')
# plt.plot(Famp_F2, z, label= 'Caso 4')
plt.plot(Famp_F10, z, label= 'Caso 5')
# plt.plot(Famp_rand, z, label= 'Caso 6')
plt.plot(Fcat, z, label='Catenaria')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()
# plt.savefig('Figuras/sol_cambioF.pdf', format='pdf')
plt.show()
```



6. $F(z) \sim \mathrm{U}(0.6, 2)$

[74]:
```
F_init = np.random.uniform(size=n-2)*1.4 + 0.6

sol_rand = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_rand = np.concatenate(([F0], sol_rand.x, [F1]))
error_rand = np.linalg.norm(Famp_rand - Fcat)/np.linalg.norm(Fcat)
```

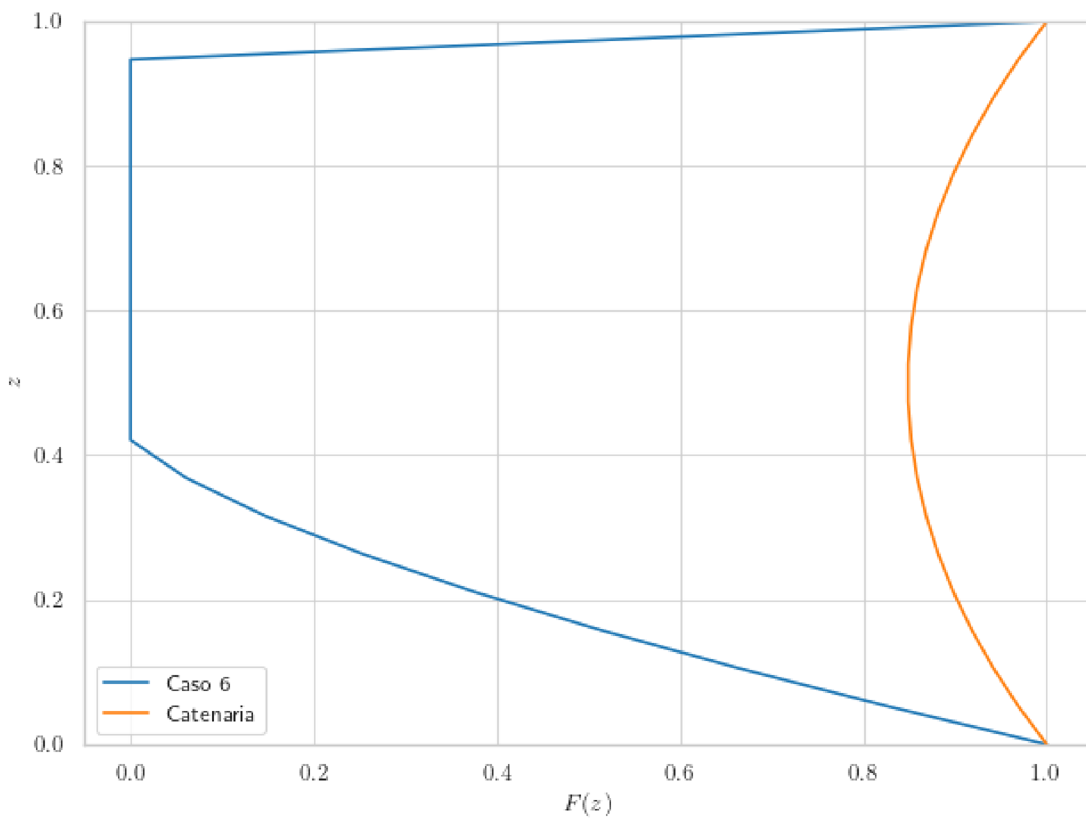
[75]:
```
print(sol_rand.message)
# print('Valor de la función:', sol_rand.fun)
# print('Resultado:', sol_rand.x)
print('Evaluaciones de la función:', sol_rand.nfev)
print('El error para F_inic=rand es:', error_rand*100)
```

```
Optimization terminated successfully
Evaluaciones de la función: 506
El error para F_inic=rand es: 80.90199087270948
```

Otra pregunta que merece la pena hacerse es qué ocurriría si se inicializan valores aleatorios dentro de la zona óptima de arranque.

Aunque existe la posibilidad de que el algoritmo converja a la solución correcta, lo cierto es que esto apenas sucede unas pocas veces y es imposible predecir en qué casos lo hará, por lo que no se considera un método robusto en el que se sepa que se va a alcanzar una solución óptima.

[76]:
```
plt.figure()
# plt.plot(Famp_F0, z, label= 'Caso 1')
# plt.plot(Famp_0, z, label= 'Caso 2')
# plt.plot(Famp_F06, z, label= 'Caso 3')
# plt.plot(Famp_F2, z, label= 'Caso 4')
# plt.plot(Famp_F10, z, label= 'Caso 5')
plt.plot(Famp_rand, z, label= 'Caso 6')
plt.plot(Fcat, z, label='Catenaria')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()
# plt.savefig('Figuras/sol_cambioF.pdf', format='pdf')
plt.show()
```



1.3.2 Minimización con un método heurístico (Differential Evolution)

El algoritmo de evolución diferencial (DE) es una técnica de optimización global que pertenece a la familia de los algoritmos evolutivos. DE se utiliza principalmente para resolver problemas de optimización con variables continuas por sus ventajas frente a los algoritmos genéticos en este campo, como es el que se trata en el problema considerado.

Principios Básicos de DE

Representación de las Soluciones

Las soluciones se representan como vectores de números reales. Cada individuo en la población es un punto en el espacio de búsqueda multidimensional.

Inicialización

La población inicial se genera de manera aleatoria dentro de los límites especificados para cada variable. Esto asegura una buena cobertura inicial del espacio de búsqueda.

Operadores Evolutivos

Mutación

Genera un nuevo vector candidato sumando la diferencia ponderada entre dos vectores a un tercer vector. Este mecanismo introduce variabilidad basada en las diferencias entre individuos de la población.

$\mathbf{v}_i = \mathbf{x}_{r1} + F \cdot (\mathbf{x}_{r2} - \mathbf{x}_{r3})$

donde $\mathbf{x}_{r1}$, $\mathbf{x}_{r2}$ y $\mathbf{x}_{r3}$ son vectores aleatorios distintos seleccionados de la población, y F es el factor de mutación que controla la amplitud de la variación diferencial.

Recombinación (Crossover)

Combina el vector mutado con el vector actual del individuo para crear un nuevo vector de prueba. Esto se realiza seleccionando componentes del vector mutado y del vector actual basándose en una tasa de recombinación.

$$\mathbf{u}_{ij} = \begin{cases} \mathbf{v}_{ij} & \text{si } \mathrm{rand}(0,1) < CR \text{ o } j = \mathrm{randint}(0, D-1) \\ \mathbf{x}_{ij} & \text{de lo contrario} \end{cases}$$

donde $\mathrm{rand}(0,1)$ es un número aleatorio entre 0 y 1, y $\mathrm{randint}(0, D-1)$ asegura que al menos un componente provenga del vector mutado.

Selección

Compara el vector de prueba con el vector actual y selecciona el mejor (el que tenga la mejor aptitud) para la siguiente generación. Esto asegura que solo las soluciones más prometedoras sobrevivan.

$ x_{i}^{(t+1)} =

$$\begin{cases} \mathbf{u}_i & \text{si } f(\mathbf{u}_i) \leq f(\mathbf{x}_i) \\ \mathbf{x}_i & \text{de lo contrario} \end{cases}$$

$

donde f es la función objetivo a minimizar.

Ventajas de Differential Evolution (DE)

Simplicidad

DE es fácil de implementar y requiere menos parámetros que muchos otros algoritmos evolutivos, como los algoritmos genéticos. Los principales parámetros son el tamaño de la población, el factor de mutación y la tasa de recombinación.

Eficiencia y Convergencia

DE es conocido por su eficiencia en la búsqueda global y su capacidad para converger a soluciones óptimas o cercanas al óptimo, incluso en espacios de búsqueda complejos con múltiples óptimos locales.

Robustez

DE es robusto y eficaz en una amplia gama de problemas de optimización, incluidos aquellos con funciones objetivo no lineales y no convexas. La combinación de mutación y recombinación basada en diferencias permite una exploración efectiva del espacio de búsqueda.

Mantenimiento de la Diversidad

La mutación diferencial basada en las diferencias entre individuos ayuda a mantener la diversidad en la población, lo que es crucial para evitar el estancamiento en óptimos locales.

Cuándo Usar Differential Evolution (DE)

Problemas de Optimización Continua

DE es especialmente adecuado para problemas donde las variables son continuas y el espacio de búsqueda es multidimensional.

Problemas No Lineales y No Convexos

Es eficaz en problemas donde la función objetivo es no lineal y no convexa, y donde otros métodos de optimización pueden quedar atrapados en óptimos locales.

Comparación con Algoritmos Genéticos (GA)

Mutación y Recombinación

- **DE**: La mutación se basa en la diferencia entre individuos, lo que ayuda a explorar el espacio de búsqueda de manera más efectiva. La recombinación en DE combina el vector mutado con el vector actual del individuo.
- **GA**: La mutación en GA suele ser aleatoria y menos dirigida. La recombinación (cruce) en GA combina partes de dos o más padres para generar nuevos individuos.

Diversidad Genética

- **DE**: Mantiene la diversidad de manera más efectiva gracias a su mecanismo de mutación diferencial, lo que ayuda a evitar el estancamiento en óptimos locales.
- **GA**: La diversidad se mantiene mediante mutación y selección, pero puede ser más propenso a quedar atrapado en óptimos locales si no se gestiona adecuadamente.

Parámetros

- **DE**: Menos parámetros a ajustar, lo que simplifica su uso y ajuste. Los principales parámetros son el factor de mutación (F), la tasa de recombinación (CR) y el tamaño de la población.
- **GA**: Más parámetros a ajustar, incluyendo tasas de cruce y mutación, tamaños de torneos de selección, y más. Esto puede hacer que GA sea más complejo de ajustar.

Aplicaciones

- **DE**: Preferido para problemas de optimización continua y de alta dimensionalidad.
- **GA**: Más versátil y adecuado para problemas combinatorios, de variables discretas, y en aplicaciones donde la representación binaria es natural.

Conclusión

Differential Evolution (DE) es una poderosa técnica de optimización global que se destaca por su simplicidad, eficiencia y robustez en una amplia gama de problemas de optimización continua. Su capacidad para mantener la diversidad en la población y evitar quedar atrapado en óptimos locales lo hace especialmente útil para problemas no lineales y no convexos. Si bien los algoritmos genéticos (GA) ofrecen versatilidad en diferentes tipos de problemas, DE proporciona una solución más dirigida y eficaz para problemas de optimización continua y de alta dimensionalidad. La elección entre DE y otros métodos de optimización debe basarse en las características específicas del problema y en los objetivos de la optimización.

Así, en este trabajo se empleará DE por su implementación más sencilla, a través de la función `scipy.optimize.differential_evolution`. Esta aproxima la solución por el método estudiado y, posteriormente, se acerca a la solución definitiva con unos pocos pasos de un algoritmo basado en gradiente.

Caso de ejemplo

[86]:
```
F_0 = 1
F0 = F_0
F1 = F_0
n = 20
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)

bounds = [(0.5, F_0) for _ in range(n-2)]
```

Clase para registrar los valores de la función objetivo.

[172]:
```
class ObjectiveLogger:
    def __init__(self):
        self.values = []
        self.iteration = 0

    def __call__(self, xk, convergence):
        value = area_func(xk)
```

```python
        self.values.append(value)
        self.iteration += 1
        print(f"Iteración: {self.iteration}, Valor de la función: {value}")
        return False

    def reset(self):
        self.values = []
        self.iteration = 0

    def log_and_reset(self, func):
        def wrapped_func(*args, **kwargs):
            self.reset()
            return func(*args, **kwargs)
        return wrapped_func

# Crear una instancia del registrador de valores
logger = ObjectiveLogger()

# Envolver differential_evolution para que resetee automáticamente el logger
differential_evolution = logger.log_and_reset(differential_evolution)
```

Ejecutar la optimización con registro de valores.

[88]:
```python
time_start = time.time()

sol_hib = differential_evolution(
    area_func,
    bounds=bounds,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=0.01,
    callback=logger,
    # polish = False
)

time_end = time.time()
time_hib = time_end - time_start
```

```
Iteración: 1, Valor de la función: 1.516848619203358
Iteración: 2, Valor de la función: 1.4352788484747168
Iteración: 3, Valor de la función: 1.22435245002439
Iteración: 4, Valor de la función: 1.22435245002439
Iteración: 5, Valor de la función: 1.22435245002439
Iteración: 6, Valor de la función: 1.1699925411404093
Iteración: 7, Valor de la función: 1.1564399294760184
```

```
Iteración: 8, Valor de la función: 1.1408680828528854
Iteración: 9, Valor de la función: 1.1408680828528854
Iteración: 10, Valor de la función: 1.1408680828528854
Iteración: 11, Valor de la función: 1.0951607787535103
Iteración: 12, Valor de la función: 1.0951607787535103
Iteración: 13, Valor de la función: 1.0951607787535103
Iteración: 14, Valor de la función: 1.0951607787535103
Iteración: 15, Valor de la función: 1.0951607787535103
Iteración: 16, Valor de la función: 1.0951607787535103
Iteración: 17, Valor de la función: 1.0951607787535103
Iteración: 18, Valor de la función: 1.0842424699430324
Iteración: 19, Valor de la función: 1.0842424699430324
Iteración: 20, Valor de la función: 1.0617893676617494
Iteración: 21, Valor de la función: 1.0612821280802733
Iteración: 22, Valor de la función: 1.0612821280802733
Iteración: 23, Valor de la función: 1.0612821280802733
Iteración: 24, Valor de la función: 1.0612821280802733
Iteración: 25, Valor de la función: 1.0612821280802733
Iteración: 26, Valor de la función: 1.0601696585793092
Iteración: 27, Valor de la función: 1.0520595057782078
Iteración: 28, Valor de la función: 1.0455596667284746
Iteración: 29, Valor de la función: 1.0399174008191685
Iteración: 30, Valor de la función: 1.0399174008191685
Iteración: 31, Valor de la función: 1.0399174008191685
Iteración: 32, Valor de la función: 1.0399174008191685
Iteración: 33, Valor de la función: 1.0399174008191685
Iteración: 34, Valor de la función: 1.0398257576243286
Iteración: 35, Valor de la función: 1.0398257576243286
Iteración: 36, Valor de la función: 1.0398257576243286
Iteración: 37, Valor de la función: 1.037962839006458
Iteración: 38, Valor de la función: 1.030966754605636
Iteración: 39, Valor de la función: 1.030966754605636
Iteración: 40, Valor de la función: 1.0254060075214653
Iteración: 41, Valor de la función: 1.0254060075214653
Iteración: 42, Valor de la función: 1.0198076586223728
Iteración: 43, Valor de la función: 1.0198076586223728
Iteración: 44, Valor de la función: 1.009699947796862
Iteración: 45, Valor de la función: 1.009699947796862
Iteración: 46, Valor de la función: 1.009699947796862
Iteración: 47, Valor de la función: 1.009699947796862
Iteración: 48, Valor de la función: 1.009699947796862
Iteración: 49, Valor de la función: 1.0080509954236319
Iteración: 50, Valor de la función: 1.0080509954236319
Iteración: 51, Valor de la función: 1.0080509954236319
Iteración: 52, Valor de la función: 1.0080509954236319
Iteración: 53, Valor de la función: 1.0080509954236319
Iteración: 54, Valor de la función: 1.0075659881228158
Iteración: 55, Valor de la función: 1.0075659881228158
```

```
Iteración: 56, Valor de la función: 1.0075659881228158
Iteración: 57, Valor de la función: 1.0075659881228158
Iteración: 58, Valor de la función: 1.0075659881228158
Iteración: 59, Valor de la función: 1.0075659881228158
Iteración: 60, Valor de la función: 1.0075659881228158
Iteración: 61, Valor de la función: 1.0075659881228158
Iteración: 62, Valor de la función: 1.0075659881228158
Iteración: 63, Valor de la función: 1.0075659881228158
Iteración: 64, Valor de la función: 1.0075659881228158
Iteración: 65, Valor de la función: 1.005644595520795
Iteración: 66, Valor de la función: 1.005644595520795
Iteración: 67, Valor de la función: 1.004569972485999
Iteración: 68, Valor de la función: 1.0019007418663115
Iteración: 69, Valor de la función: 1.0019007418663115
Iteración: 70, Valor de la función: 0.99866070295518
Iteración: 71, Valor de la función: 0.99866070295518
Iteración: 72, Valor de la función: 0.99866070295518
Iteración: 73, Valor de la función: 0.9959063776158852
Iteración: 74, Valor de la función: 0.9959063776158852
Iteración: 75, Valor de la función: 0.9959063776158852
Iteración: 76, Valor de la función: 0.9959063776158852
Iteración: 77, Valor de la función: 0.9959063776158852
Iteración: 78, Valor de la función: 0.9959063776158852
Iteración: 79, Valor de la función: 0.9959063776158852
Iteración: 80, Valor de la función: 0.9959063776158852
Iteración: 81, Valor de la función: 0.9959063776158852
Iteración: 82, Valor de la función: 0.9959063776158852
Iteración: 83, Valor de la función: 0.9959063776158852
Iteración: 84, Valor de la función: 0.9896896851158962
Iteración: 85, Valor de la función: 0.9896896851158962
Iteración: 86, Valor de la función: 0.9896896851158962
Iteración: 87, Valor de la función: 0.9896896851158962
Iteración: 88, Valor de la función: 0.9896896851158962
Iteración: 89, Valor de la función: 0.9888769996754097
Iteración: 90, Valor de la función: 0.9887278835192754
Iteración: 91, Valor de la función: 0.9887278835192754
Iteración: 92, Valor de la función: 0.9887278835192754
Iteración: 93, Valor de la función: 0.9887278835192754
Iteración: 94, Valor de la función: 0.9867813663798363
Iteración: 95, Valor de la función: 0.9867813663798363
Iteración: 96, Valor de la función: 0.9867813663798363
Iteración: 97, Valor de la función: 0.9844579028568966
Iteración: 98, Valor de la función: 0.9844579028568966
Iteración: 99, Valor de la función: 0.981180232568267
Iteración: 100, Valor de la función: 0.9783076970534758
Iteración: 101, Valor de la función: 0.9783076970534758
Iteración: 102, Valor de la función: 0.9783076970534758
Iteración: 103, Valor de la función: 0.9783076970534758
```

```
Iteración: 104, Valor de la función: 0.97251748773522
Iteración: 105, Valor de la función: 0.97251748773522
Iteración: 106, Valor de la función: 0.97251748773522
Iteración: 107, Valor de la función: 0.97251748773522
Iteración: 108, Valor de la función: 0.97251748773522
Iteración: 109, Valor de la función: 0.97251748773522
Iteración: 110, Valor de la función: 0.97251748773522
Iteración: 111, Valor de la función: 0.97251748773522
Iteración: 112, Valor de la función: 0.97251748773522
Iteración: 113, Valor de la función: 0.97251748773522
Iteración: 114, Valor de la función: 0.97251748773522
Iteración: 115, Valor de la función: 0.97251748773522
Iteración: 116, Valor de la función: 0.97251748773522
Iteración: 117, Valor de la función: 0.97251748773522
Iteración: 118, Valor de la función: 0.97251748773522
Iteración: 119, Valor de la función: 0.97251748773522
Iteración: 120, Valor de la función: 0.97251748773522
Iteración: 121, Valor de la función: 0.97251748773522
Iteración: 122, Valor de la función: 0.97251748773522
Iteración: 123, Valor de la función: 0.9662075854198217
Iteración: 124, Valor de la función: 0.9662075854198217
Iteración: 125, Valor de la función: 0.9662075854198217
Iteración: 126, Valor de la función: 0.9662075854198217
Iteración: 127, Valor de la función: 0.9662075854198217
Iteración: 128, Valor de la función: 0.9662075854198217
Iteración: 129, Valor de la función: 0.9662075854198217
Iteración: 130, Valor de la función: 0.9662075854198217
Iteración: 131, Valor de la función: 0.9662075854198217
Iteración: 132, Valor de la función: 0.9662075854198217
Iteración: 133, Valor de la función: 0.9662075854198217
Iteración: 134, Valor de la función: 0.9662075854198217
Iteración: 135, Valor de la función: 0.9662075854198217
Iteración: 136, Valor de la función: 0.9662075854198217
Iteración: 137, Valor de la función: 0.9662075854198217
Iteración: 138, Valor de la función: 0.9662075854198217
Iteración: 139, Valor de la función: 0.9662075854198217
Iteración: 140, Valor de la función: 0.9662075854198217
Iteración: 141, Valor de la función: 0.9662075854198217
Iteración: 142, Valor de la función: 0.9662075854198217
Iteración: 143, Valor de la función: 0.9662075854198217
Iteración: 144, Valor de la función: 0.9662075854198217
Iteración: 145, Valor de la función: 0.9662075854198217
Iteración: 146, Valor de la función: 0.9662075854198217
Iteración: 147, Valor de la función: 0.9662075854198217
Iteración: 148, Valor de la función: 0.9662075854198217
Iteración: 149, Valor de la función: 0.9662075854198217
Iteración: 150, Valor de la función: 0.9662075854198217
Iteración: 151, Valor de la función: 0.9662075854198217
```

```
Iteración: 152, Valor de la función: 0.9662075854198217
Iteración: 153, Valor de la función: 0.9662075854198217
Iteración: 154, Valor de la función: 0.9662075854198217
Iteración: 155, Valor de la función: 0.9662075854198217
Iteración: 156, Valor de la función: 0.9662075854198217
Iteración: 157, Valor de la función: 0.9662075854198217
Iteración: 158, Valor de la función: 0.9662075854198217
Iteración: 159, Valor de la función: 0.9662075854198217
Iteración: 160, Valor de la función: 0.9662075854198217
Iteración: 161, Valor de la función: 0.9662075854198217
Iteración: 162, Valor de la función: 0.9662075854198217
Iteración: 163, Valor de la función: 0.9662075854198217
Iteración: 164, Valor de la función: 0.9662075854198217
Iteración: 165, Valor de la función: 0.9662075854198217
Iteración: 166, Valor de la función: 0.9662075854198217
Iteración: 167, Valor de la función: 0.9662075854198217
Iteración: 168, Valor de la función: 0.9662075854198217
Iteración: 169, Valor de la función: 0.9662075854198217
Iteración: 170, Valor de la función: 0.9662075854198217
Iteración: 171, Valor de la función: 0.9662075854198217
Iteración: 172, Valor de la función: 0.9662075854198217
Iteración: 173, Valor de la función: 0.9662075854198217
Iteración: 174, Valor de la función: 0.9662075854198217
Iteración: 175, Valor de la función: 0.9662075854198217
Iteración: 176, Valor de la función: 0.9662075854198217
Iteración: 177, Valor de la función: 0.9662075854198217
Iteración: 178, Valor de la función: 0.9662075854198217
Iteración: 179, Valor de la función: 0.9662075854198217
Iteración: 180, Valor de la función: 0.9662075854198217
Iteración: 181, Valor de la función: 0.9662075854198217
Iteración: 182, Valor de la función: 0.9662075854198217
Iteración: 183, Valor de la función: 0.9662075854198217
Iteración: 184, Valor de la función: 0.9662075854198217
Iteración: 185, Valor de la función: 0.9662075854198217
Iteración: 186, Valor de la función: 0.9662075854198217
Iteración: 187, Valor de la función: 0.9662075854198217
Iteración: 188, Valor de la función: 0.9662075854198217
Iteración: 189, Valor de la función: 0.9662075854198217
Iteración: 190, Valor de la función: 0.9662075854198217
Iteración: 191, Valor de la función: 0.9662075854198217
Iteración: 192, Valor de la función: 0.9662075854198217
Iteración: 193, Valor de la función: 0.9662075854198217
Iteración: 194, Valor de la función: 0.9656876392815109
Iteración: 195, Valor de la función: 0.9656876392815109
Iteración: 196, Valor de la función: 0.9656876392815109
Iteración: 197, Valor de la función: 0.9656876392815109
Iteración: 198, Valor de la función: 0.9656876392815109
Iteración: 199, Valor de la función: 0.9656876392815109
```

```
Iteración: 200, Valor de la función: 0.9656876392815109
Iteración: 201, Valor de la función: 0.9656876392815109
Iteración: 202, Valor de la función: 0.9656876392815109
Iteración: 203, Valor de la función: 0.9656876392815109
Iteración: 204, Valor de la función: 0.9656876392815109
Iteración: 205, Valor de la función: 0.9656876392815109
Iteración: 206, Valor de la función: 0.9656876392815109
Iteración: 207, Valor de la función: 0.9656876392815109
Iteración: 208, Valor de la función: 0.9656876392815109
Iteración: 209, Valor de la función: 0.9656876392815109
Iteración: 210, Valor de la función: 0.9656876392815109
Iteración: 211, Valor de la función: 0.9656876392815109
Iteración: 212, Valor de la función: 0.9656876392815109
Iteración: 213, Valor de la función: 0.9656876392815109
Iteración: 214, Valor de la función: 0.9656876392815109
Iteración: 215, Valor de la función: 0.9656876392815109
Iteración: 216, Valor de la función: 0.9656876392815109
Iteración: 217, Valor de la función: 0.9656876392815109
Iteración: 218, Valor de la función: 0.9656876392815109
Iteración: 219, Valor de la función: 0.9656876392815109
Iteración: 220, Valor de la función: 0.9656876392815109
Iteración: 221, Valor de la función: 0.9656876392815109
Iteración: 222, Valor de la función: 0.9656876392815109
Iteración: 223, Valor de la función: 0.9656876392815109
Iteración: 224, Valor de la función: 0.9656876392815109
Iteración: 225, Valor de la función: 0.9656876392815109
Iteración: 226, Valor de la función: 0.9656876392815109
Iteración: 227, Valor de la función: 0.9656876392815109
Iteración: 228, Valor de la función: 0.9656876392815109
Iteración: 229, Valor de la función: 0.9656876392815109
Iteración: 230, Valor de la función: 0.9656876392815109
Iteración: 231, Valor de la función: 0.9656876392815109
Iteración: 232, Valor de la función: 0.9656876392815109
Iteración: 233, Valor de la función: 0.9656876392815109
Iteración: 234, Valor de la función: 0.9656876392815109
Iteración: 235, Valor de la función: 0.9656876392815109
Iteración: 236, Valor de la función: 0.9654062986724368
Iteración: 237, Valor de la función: 0.9654062986724368
Iteración: 238, Valor de la función: 0.9654062986724368
Iteración: 239, Valor de la función: 0.9654062986724368
Iteración: 240, Valor de la función: 0.9654062986724368
Iteración: 241, Valor de la función: 0.9654062986724368
Iteración: 242, Valor de la función: 0.9632603633247343
Iteración: 243, Valor de la función: 0.9632603633247343
Iteración: 244, Valor de la función: 0.9632603633247343
Iteración: 245, Valor de la función: 0.9632603633247343
Iteración: 246, Valor de la función: 0.9632603633247343
Iteración: 247, Valor de la función: 0.9632603633247343
```

```
Iteración: 248, Valor de la función: 0.9632603633247343
Iteración: 249, Valor de la función: 0.9632603633247343
Iteración: 250, Valor de la función: 0.9632603633247343
Iteración: 251, Valor de la función: 0.9632603633247343
Iteración: 252, Valor de la función: 0.9625710769782498
Iteración: 253, Valor de la función: 0.9625710769782498
Iteración: 254, Valor de la función: 0.9625710769782498
Iteración: 255, Valor de la función: 0.9625710769782498
Iteración: 256, Valor de la función: 0.9625710769782498
Iteración: 257, Valor de la función: 0.9625710769782498
Iteración: 258, Valor de la función: 0.9625710769782498
Iteración: 259, Valor de la función: 0.9625710769782498
Iteración: 260, Valor de la función: 0.9625710769782498
Iteración: 261, Valor de la función: 0.9625710769782498
Iteración: 262, Valor de la función: 0.9625710769782498
Iteración: 263, Valor de la función: 0.9625710769782498
Iteración: 264, Valor de la función: 0.9622447922321269
Iteración: 265, Valor de la función: 0.9622447922321269
Iteración: 266, Valor de la función: 0.9622447922321269
Iteración: 267, Valor de la función: 0.9622447922321269
Iteración: 268, Valor de la función: 0.9622447922321269
Iteración: 269, Valor de la función: 0.9622447922321269
Iteración: 270, Valor de la función: 0.9622447922321269
Iteración: 271, Valor de la función: 0.9622447922321269
Iteración: 272, Valor de la función: 0.9622447922321269
Iteración: 273, Valor de la función: 0.9622447922321269
Iteración: 274, Valor de la función: 0.9615570403533349
Iteración: 275, Valor de la función: 0.9615570403533349
Iteración: 276, Valor de la función: 0.9615570403533349
Iteración: 277, Valor de la función: 0.9615570403533349
Iteración: 278, Valor de la función: 0.9615570403533349
Iteración: 279, Valor de la función: 0.9615570403533349
Iteración: 280, Valor de la función: 0.9615570403533349
Iteración: 281, Valor de la función: 0.9615570403533349
Iteración: 282, Valor de la función: 0.9615570403533349
Iteración: 283, Valor de la función: 0.9615570403533349
Iteración: 284, Valor de la función: 0.9615570403533349
Iteración: 285, Valor de la función: 0.9615570403533349
Iteración: 286, Valor de la función: 0.9615570403533349
Iteración: 287, Valor de la función: 0.9615570403533349
Iteración: 288, Valor de la función: 0.9615570403533349
Iteración: 289, Valor de la función: 0.9615570403533349
Iteración: 290, Valor de la función: 0.9615570403533349
Iteración: 291, Valor de la función: 0.9615570403533349
Iteración: 292, Valor de la función: 0.9615570403533349
Iteración: 293, Valor de la función: 0.9615570403533349
Iteración: 294, Valor de la función: 0.9615570403533349
Iteración: 295, Valor de la función: 0.9615570403533349
```

```
Iteración: 296, Valor de la función: 0.9615570403533349
Iteración: 297, Valor de la función: 0.9615570403533349
Iteración: 298, Valor de la función: 0.9615570403533349
Iteración: 299, Valor de la función: 0.9615570403533349
Iteración: 300, Valor de la función: 0.9615570403533349
Iteración: 301, Valor de la función: 0.9615570403533349
Iteración: 302, Valor de la función: 0.9615570403533349
Iteración: 303, Valor de la función: 0.9615570403533349
Iteración: 304, Valor de la función: 0.9615570403533349
Iteración: 305, Valor de la función: 0.9615570403533349
Iteración: 306, Valor de la función: 0.9607205408892311
Iteración: 307, Valor de la función: 0.9607205408892311
Iteración: 308, Valor de la función: 0.9579644376194785
Iteración: 309, Valor de la función: 0.9579644376194785
Iteración: 310, Valor de la función: 0.9579644376194785
Iteración: 311, Valor de la función: 0.9579644376194785
Iteración: 312, Valor de la función: 0.9579644376194785
Iteración: 313, Valor de la función: 0.9579644376194785
Iteración: 314, Valor de la función: 0.9579644376194785
Iteración: 315, Valor de la función: 0.9579644376194785
Iteración: 316, Valor de la función: 0.9579644376194785
Iteración: 317, Valor de la función: 0.9579644376194785
Iteración: 318, Valor de la función: 0.9579644376194785
Iteración: 319, Valor de la función: 0.9579644376194785
Iteración: 320, Valor de la función: 0.9579644376194785
Iteración: 321, Valor de la función: 0.9579644376194785
Iteración: 322, Valor de la función: 0.9579644376194785
Iteración: 323, Valor de la función: 0.9579644376194785
Iteración: 324, Valor de la función: 0.9579644376194785
Iteración: 325, Valor de la función: 0.9579644376194785
Iteración: 326, Valor de la función: 0.9579644376194785
Iteración: 327, Valor de la función: 0.9579644376194785
Iteración: 328, Valor de la función: 0.9579644376194785
Iteración: 329, Valor de la función: 0.9579644376194785
Iteración: 330, Valor de la función: 0.9579644376194785
Iteración: 331, Valor de la función: 0.9579644376194785
Iteración: 332, Valor de la función: 0.9579644376194785
Iteración: 333, Valor de la función: 0.9579644376194785
Iteración: 334, Valor de la función: 0.9579644376194785
Iteración: 335, Valor de la función: 0.9579644376194785
Iteración: 336, Valor de la función: 0.9579644376194785
Iteración: 337, Valor de la función: 0.9579644376194785
Iteración: 338, Valor de la función: 0.9579644376194785
Iteración: 339, Valor de la función: 0.9579644376194785
Iteración: 340, Valor de la función: 0.9579644376194785
Iteración: 341, Valor de la función: 0.9579644376194785
Iteración: 342, Valor de la función: 0.9579644376194785
Iteración: 343, Valor de la función: 0.9579644376194785
```

```
Iteración: 344, Valor de la función: 0.9579644376194785
Iteración: 345, Valor de la función: 0.9579644376194785
Iteración: 346, Valor de la función: 0.9579644376194785
Iteración: 347, Valor de la función: 0.9579644376194785
Iteración: 348, Valor de la función: 0.9579644376194785
Iteración: 349, Valor de la función: 0.9579644376194785
Iteración: 350, Valor de la función: 0.9579644376194785
Iteración: 351, Valor de la función: 0.9579644376194785
Iteración: 352, Valor de la función: 0.9579644376194785
Iteración: 353, Valor de la función: 0.9579644376194785
Iteración: 354, Valor de la función: 0.9579644376194785
Iteración: 355, Valor de la función: 0.9579644376194785
Iteración: 356, Valor de la función: 0.9579644376194785
Iteración: 357, Valor de la función: 0.9579644376194785
Iteración: 358, Valor de la función: 0.9579644376194785
Iteración: 359, Valor de la función: 0.9579644376194785
Iteración: 360, Valor de la función: 0.9579644376194785
Iteración: 361, Valor de la función: 0.9579644376194785
Iteración: 362, Valor de la función: 0.9579644376194785
Iteración: 363, Valor de la función: 0.9579644376194785
Iteración: 364, Valor de la función: 0.9579644376194785
Iteración: 365, Valor de la función: 0.9579644376194785
Iteración: 366, Valor de la función: 0.9579644376194785
Iteración: 367, Valor de la función: 0.9579644376194785
Iteración: 368, Valor de la función: 0.9579644376194785
Iteración: 369, Valor de la función: 0.9579644376194785
Iteración: 370, Valor de la función: 0.9579644376194785
Iteración: 371, Valor de la función: 0.9579644376194785
Iteración: 372, Valor de la función: 0.9579644376194785
Iteración: 373, Valor de la función: 0.9579644376194785
Iteración: 374, Valor de la función: 0.9579644376194785
Iteración: 375, Valor de la función: 0.9579644376194785
Iteración: 376, Valor de la función: 0.9579644376194785
Iteración: 377, Valor de la función: 0.9579644376194785
Iteración: 378, Valor de la función: 0.9564037349604503
Iteración: 379, Valor de la función: 0.9564037349604503
Iteración: 380, Valor de la función: 0.9564037349604503
Iteración: 381, Valor de la función: 0.9564037349604503
Iteración: 382, Valor de la función: 0.9564037349604503
Iteración: 383, Valor de la función: 0.9564037349604503
Iteración: 384, Valor de la función: 0.9564037349604503
Iteración: 385, Valor de la función: 0.9564037349604503
Iteración: 386, Valor de la función: 0.9564037349604503
Iteración: 387, Valor de la función: 0.9564037349604503
Iteración: 388, Valor de la función: 0.9564037349604503
Iteración: 389, Valor de la función: 0.9564037349604503
Iteración: 390, Valor de la función: 0.9564037349604503
Iteración: 391, Valor de la función: 0.9564037349604503
```

```
Iteración: 392, Valor de la función: 0.9564037349604503
Iteración: 393, Valor de la función: 0.9564037349604503
Iteración: 394, Valor de la función: 0.9564037349604503
Iteración: 395, Valor de la función: 0.9564037349604503
Iteración: 396, Valor de la función: 0.9564037349604503
Iteración: 397, Valor de la función: 0.9564037349604503
Iteración: 398, Valor de la función: 0.9564037349604503
Iteración: 399, Valor de la función: 0.9564037349604503
Iteración: 400, Valor de la función: 0.9564037349604503
Iteración: 401, Valor de la función: 0.9564037349604503
Iteración: 402, Valor de la función: 0.9564037349604503
Iteración: 403, Valor de la función: 0.9564037349604503
Iteración: 404, Valor de la función: 0.9564037349604503
Iteración: 405, Valor de la función: 0.9564037349604503
Iteración: 406, Valor de la función: 0.9564037349604503
Iteración: 407, Valor de la función: 0.9564037349604503
Iteración: 408, Valor de la función: 0.9564037349604503
Iteración: 409, Valor de la función: 0.9564037349604503
Iteración: 410, Valor de la función: 0.9564037349604503
Iteración: 411, Valor de la función: 0.9564037349604503
Iteración: 412, Valor de la función: 0.9564037349604503
Iteración: 413, Valor de la función: 0.9564037349604503
Iteración: 414, Valor de la función: 0.9564037349604503
Iteración: 415, Valor de la función: 0.9564037349604503
Iteración: 416, Valor de la función: 0.9564037349604503
Iteración: 417, Valor de la función: 0.9563584414536657
Iteración: 418, Valor de la función: 0.9563584414536657
Iteración: 419, Valor de la función: 0.9563584414536657
Iteración: 420, Valor de la función: 0.9563584414536657
Iteración: 421, Valor de la función: 0.9563584414536657
Iteración: 422, Valor de la función: 0.9563584414536657
Iteración: 423, Valor de la función: 0.9563584414536657
Iteración: 424, Valor de la función: 0.9563584414536657
Iteración: 425, Valor de la función: 0.9563584414536657
Iteración: 426, Valor de la función: 0.9563584414536657
Iteración: 427, Valor de la función: 0.9563584414536657
Iteración: 428, Valor de la función: 0.9563584414536657
Iteración: 429, Valor de la función: 0.9563584414536657
Iteración: 430, Valor de la función: 0.9563584414536657
Iteración: 431, Valor de la función: 0.9561348558217816
Iteración: 432, Valor de la función: 0.9561348558217816
Iteración: 433, Valor de la función: 0.955703350553775
Iteración: 434, Valor de la función: 0.955703350553775
Iteración: 435, Valor de la función: 0.955703350553775
Iteración: 436, Valor de la función: 0.955703350553775
Iteración: 437, Valor de la función: 0.9553886044239215
Iteración: 438, Valor de la función: 0.9553886044239215
Iteración: 439, Valor de la función: 0.9553886044239215
```

```
Iteración: 440, Valor de la función: 0.9553886044239215
Iteración: 441, Valor de la función: 0.9553886044239215
Iteración: 442, Valor de la función: 0.9553886044239215
Iteración: 443, Valor de la función: 0.9553886044239215
Iteración: 444, Valor de la función: 0.9553886044239215
Iteración: 445, Valor de la función: 0.9553886044239215
Iteración: 446, Valor de la función: 0.9553886044239215
Iteración: 447, Valor de la función: 0.9553886044239215
Iteración: 448, Valor de la función: 0.9553886044239215
Iteración: 449, Valor de la función: 0.9553886044239215
Iteración: 450, Valor de la función: 0.9553886044239215
Iteración: 451, Valor de la función: 0.9553886044239215
Iteración: 452, Valor de la función: 0.9553886044239215
Iteración: 453, Valor de la función: 0.9553886044239215
Iteración: 454, Valor de la función: 0.9553886044239215
Iteración: 455, Valor de la función: 0.9553886044239215
Iteración: 456, Valor de la función: 0.9553886044239215
Iteración: 457, Valor de la función: 0.9553886044239215
Iteración: 458, Valor de la función: 0.9553886044239215
Iteración: 459, Valor de la función: 0.9553886044239215
Iteración: 460, Valor de la función: 0.9553886044239215
Iteración: 461, Valor de la función: 0.9553886044239215
Iteración: 462, Valor de la función: 0.9553886044239215
Iteración: 463, Valor de la función: 0.9553886044239215
Iteración: 464, Valor de la función: 0.9553886044239215
Iteración: 465, Valor de la función: 0.9553886044239215
Iteración: 466, Valor de la función: 0.9553886044239215
Iteración: 467, Valor de la función: 0.9553886044239215
Iteración: 468, Valor de la función: 0.9553886044239215
Iteración: 469, Valor de la función: 0.9553886044239215
Iteración: 470, Valor de la función: 0.9553886044239215
Iteración: 471, Valor de la función: 0.9553886044239215
Iteración: 472, Valor de la función: 0.9553886044239215
Iteración: 473, Valor de la función: 0.9553886044239215
Iteración: 474, Valor de la función: 0.955286729953442
Iteración: 475, Valor de la función: 0.955286729953442
Iteración: 476, Valor de la función: 0.955286729953442
Iteración: 477, Valor de la función: 0.955286729953442
Iteración: 478, Valor de la función: 0.955286729953442
Iteración: 479, Valor de la función: 0.9552139916445298
Iteración: 480, Valor de la función: 0.9552139916445298
Iteración: 481, Valor de la función: 0.9552139916445298
Iteración: 482, Valor de la función: 0.9552139916445298
Iteración: 483, Valor de la función: 0.9552139916445298
Iteración: 484, Valor de la función: 0.9552139916445298
Iteración: 485, Valor de la función: 0.9552139916445298
Iteración: 486, Valor de la función: 0.9552139916445298
Iteración: 487, Valor de la función: 0.9552139916445298
```

```
Iteración: 488, Valor de la función: 0.9552139916445298
Iteración: 489, Valor de la función: 0.9552139916445298
Iteración: 490, Valor de la función: 0.9552139916445298
Iteración: 491, Valor de la función: 0.9552139916445298
Iteración: 492, Valor de la función: 0.9552139916445298
Iteración: 493, Valor de la función: 0.9552139916445298
Iteración: 494, Valor de la función: 0.9552139916445298
Iteración: 495, Valor de la función: 0.9552139916445298
Iteración: 496, Valor de la función: 0.9552139916445298
Iteración: 497, Valor de la función: 0.9552139916445298
Iteración: 498, Valor de la función: 0.9548862316744547
Iteración: 499, Valor de la función: 0.9548862316744547
Iteración: 500, Valor de la función: 0.9548862316744547
Iteración: 501, Valor de la función: 0.9548862316744547
Iteración: 502, Valor de la función: 0.9548862316744547
Iteración: 503, Valor de la función: 0.9548862316744547
Iteración: 504, Valor de la función: 0.9548862316744547
Iteración: 505, Valor de la función: 0.9548862316744547
Iteración: 506, Valor de la función: 0.9548862316744547
Iteración: 507, Valor de la función: 0.9548862316744547
Iteración: 508, Valor de la función: 0.9548862316744547
Iteración: 509, Valor de la función: 0.9548862316744547
Iteración: 510, Valor de la función: 0.9548862316744547
Iteración: 511, Valor de la función: 0.9548862316744547
Iteración: 512, Valor de la función: 0.9548862316744547
Iteración: 513, Valor de la función: 0.9548862316744547
Iteración: 514, Valor de la función: 0.9548862316744547
Iteración: 515, Valor de la función: 0.9548862316744547
Iteración: 516, Valor de la función: 0.9548811953584759
Iteración: 517, Valor de la función: 0.9548811953584759
Iteración: 518, Valor de la función: 0.9548811953584759
Iteración: 519, Valor de la función: 0.9548323175119147
Iteración: 520, Valor de la función: 0.9548323175119147
Iteración: 521, Valor de la función: 0.9548323175119147
Iteración: 522, Valor de la función: 0.9548323175119147
Iteración: 523, Valor de la función: 0.9548323175119147
Iteración: 524, Valor de la función: 0.9548323175119147
Iteración: 525, Valor de la función: 0.9548323175119147
Iteración: 526, Valor de la función: 0.9548323175119147
Iteración: 527, Valor de la función: 0.9548323175119147
Iteración: 528, Valor de la función: 0.9548323175119147
Iteración: 529, Valor de la función: 0.9548323175119147
Iteración: 530, Valor de la función: 0.9548323175119147
Iteración: 531, Valor de la función: 0.9548323175119147
Iteración: 532, Valor de la función: 0.9548323175119147
Iteración: 533, Valor de la función: 0.9548323175119147
Iteración: 534, Valor de la función: 0.9548323175119147
Iteración: 535, Valor de la función: 0.9548130057406169
```

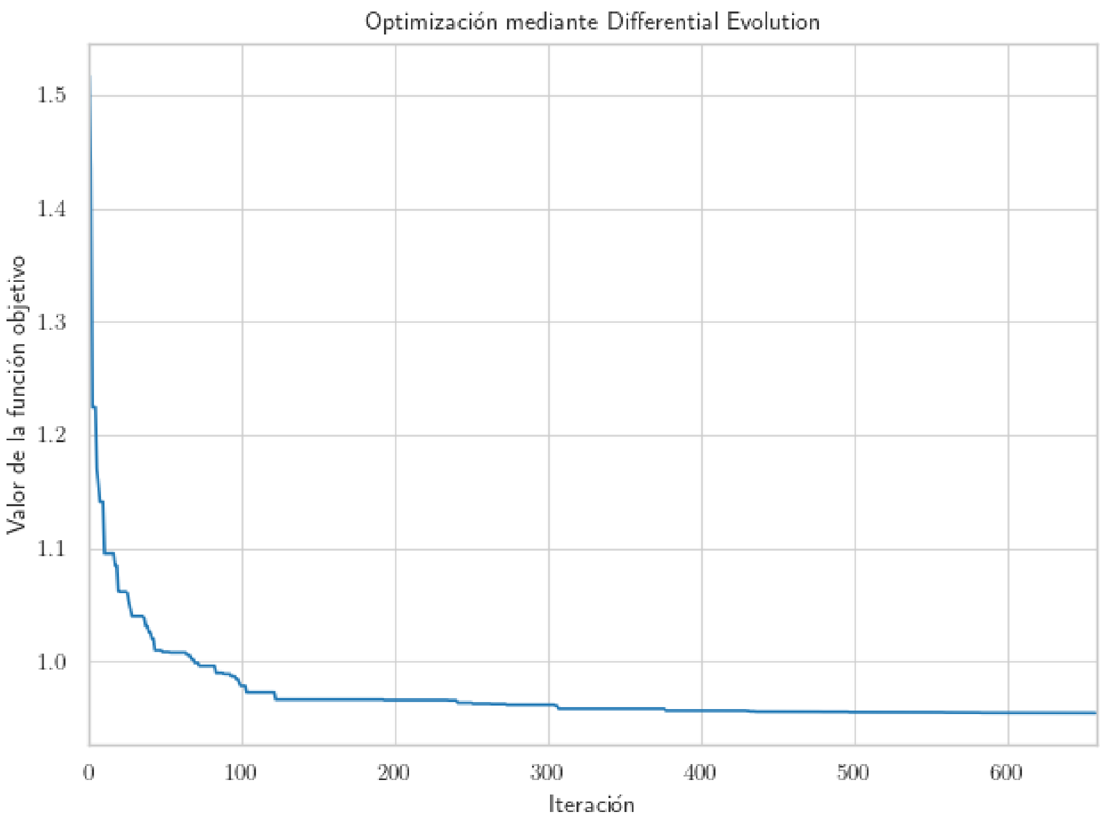
```
Iteración: 536, Valor de la función: 0.9548130057406169
Iteración: 537, Valor de la función: 0.9548130057406169
Iteración: 538, Valor de la función: 0.9548130057406169
Iteración: 539, Valor de la función: 0.9548130057406169
Iteración: 540, Valor de la función: 0.9548130057406169
Iteración: 541, Valor de la función: 0.9548130057406169
Iteración: 542, Valor de la función: 0.9548130057406169
Iteración: 543, Valor de la función: 0.9548130057406169
Iteración: 544, Valor de la función: 0.9548130057406169
Iteración: 545, Valor de la función: 0.9548130057406169
Iteración: 546, Valor de la función: 0.9548130057406169
Iteración: 547, Valor de la función: 0.9548130057406169
Iteración: 548, Valor de la función: 0.954793904247825
Iteración: 549, Valor de la función: 0.9547163799011122
Iteración: 550, Valor de la función: 0.9547163799011122
Iteración: 551, Valor de la función: 0.9547163799011122
Iteración: 552, Valor de la función: 0.9547163799011122
Iteración: 553, Valor de la función: 0.9547163799011122
Iteración: 554, Valor de la función: 0.9547163799011122
Iteración: 555, Valor de la función: 0.9547163799011122
Iteración: 556, Valor de la función: 0.9547163799011122
Iteración: 557, Valor de la función: 0.9547163799011122
Iteración: 558, Valor de la función: 0.9547163799011122
Iteración: 559, Valor de la función: 0.9547163799011122
Iteración: 560, Valor de la función: 0.9545646382176305
Iteración: 561, Valor de la función: 0.9545646382176305
Iteración: 562, Valor de la función: 0.9545646382176305
Iteración: 563, Valor de la función: 0.9545646382176305
Iteración: 564, Valor de la función: 0.9545646382176305
Iteración: 565, Valor de la función: 0.9545646382176305
Iteración: 566, Valor de la función: 0.9545646382176305
Iteración: 567, Valor de la función: 0.9545646382176305
Iteración: 568, Valor de la función: 0.9545646382176305
Iteración: 569, Valor de la función: 0.9545646382176305
Iteración: 570, Valor de la función: 0.9545646382176305
Iteración: 571, Valor de la función: 0.9545646382176305
Iteración: 572, Valor de la función: 0.9545646382176305
Iteración: 573, Valor de la función: 0.9544454940454153
Iteración: 574, Valor de la función: 0.9544454940454153
Iteración: 575, Valor de la función: 0.9544454940454153
Iteración: 576, Valor de la función: 0.9544454940454153
Iteración: 577, Valor de la función: 0.9544454940454153
Iteración: 578, Valor de la función: 0.9544454940454153
Iteración: 579, Valor de la función: 0.9544454940454153
Iteración: 580, Valor de la función: 0.9544454940454153
Iteración: 581, Valor de la función: 0.9544454940454153
Iteración: 582, Valor de la función: 0.9544454940454153
Iteración: 583, Valor de la función: 0.9544454940454153
```

```
Iteración: 584, Valor de la función: 0.9542990836075539
Iteración: 585, Valor de la función: 0.9542990836075539
Iteración: 586, Valor de la función: 0.9542990836075539
Iteración: 587, Valor de la función: 0.9542990836075539
Iteración: 588, Valor de la función: 0.9542990836075539
Iteración: 589, Valor de la función: 0.9542990836075539
Iteración: 590, Valor de la función: 0.9542990836075539
Iteración: 591, Valor de la función: 0.9542990836075539
Iteración: 592, Valor de la función: 0.9542990836075539
Iteración: 593, Valor de la función: 0.9542990836075539
Iteración: 594, Valor de la función: 0.9542990836075539
Iteración: 595, Valor de la función: 0.9542990836075539
Iteración: 596, Valor de la función: 0.9542990836075539
Iteración: 597, Valor de la función: 0.9542990836075539
Iteración: 598, Valor de la función: 0.9542990836075539
Iteración: 599, Valor de la función: 0.9542990836075539
Iteración: 600, Valor de la función: 0.9542990836075539
Iteración: 601, Valor de la función: 0.9542990836075539
Iteración: 602, Valor de la función: 0.9542990836075539
Iteración: 603, Valor de la función: 0.9542990836075539
Iteración: 604, Valor de la función: 0.9542990836075539
Iteración: 605, Valor de la función: 0.9542990836075539
Iteración: 606, Valor de la función: 0.9542990836075539
Iteración: 607, Valor de la función: 0.9542990836075539
Iteración: 608, Valor de la función: 0.9542990836075539
Iteración: 609, Valor de la función: 0.9542990836075539
Iteración: 610, Valor de la función: 0.9542990836075539
Iteración: 611, Valor de la función: 0.9542990836075539
Iteración: 612, Valor de la función: 0.9542990836075539
Iteración: 613, Valor de la función: 0.9542990836075539
Iteración: 614, Valor de la función: 0.9542990836075539
Iteración: 615, Valor de la función: 0.9542990836075539
Iteración: 616, Valor de la función: 0.9542990836075539
Iteración: 617, Valor de la función: 0.9542990836075539
Iteración: 618, Valor de la función: 0.9542390821231378
Iteración: 619, Valor de la función: 0.9542390821231378
Iteración: 620, Valor de la función: 0.9542390821231378
Iteración: 621, Valor de la función: 0.9542390821231378
Iteración: 622, Valor de la función: 0.9542390821231378
Iteración: 623, Valor de la función: 0.9542390821231378
Iteración: 624, Valor de la función: 0.9542390821231378
Iteración: 625, Valor de la función: 0.9542390821231378
Iteración: 626, Valor de la función: 0.9542390821231378
Iteración: 627, Valor de la función: 0.9542390821231378
Iteración: 628, Valor de la función: 0.9542390821231378
Iteración: 629, Valor de la función: 0.9542390821231378
Iteración: 630, Valor de la función: 0.9542390821231378
Iteración: 631, Valor de la función: 0.9542390821231378
```

```
Iteración: 632, Valor de la función: 0.9542390821231378
Iteración: 633, Valor de la función: 0.9542390821231378
Iteración: 634, Valor de la función: 0.9542390821231378
Iteración: 635, Valor de la función: 0.9542197177749537
Iteración: 636, Valor de la función: 0.9542197177749537
Iteración: 637, Valor de la función: 0.9542197177749537
Iteración: 638, Valor de la función: 0.9542197177749537
Iteración: 639, Valor de la función: 0.9542197177749537
Iteración: 640, Valor de la función: 0.9542197177749537
Iteración: 641, Valor de la función: 0.9542197177749537
Iteración: 642, Valor de la función: 0.9542197177749537
Iteración: 643, Valor de la función: 0.9542197177749537
Iteración: 644, Valor de la función: 0.9542154435435085
Iteración: 645, Valor de la función: 0.9542154435435085
Iteración: 646, Valor de la función: 0.9542128043592976
Iteración: 647, Valor de la función: 0.9542128043592976
Iteración: 648, Valor de la función: 0.9542128043592976
Iteración: 649, Valor de la función: 0.9542128043592976
Iteración: 650, Valor de la función: 0.9541672913411235
Iteración: 651, Valor de la función: 0.9541672913411235
Iteración: 652, Valor de la función: 0.9541672913411235
Iteración: 653, Valor de la función: 0.9541672913411235
Iteración: 654, Valor de la función: 0.9541672913411235
Iteración: 655, Valor de la función: 0.9541672913411235
Iteración: 656, Valor de la función: 0.9541672913411235
Iteración: 657, Valor de la función: 0.9541073787587865
Iteración: 658, Valor de la función: 0.9541073787587865
Iteración: 659, Valor de la función: 0.9541073787587865
```

Representación gráfica de la evolución de la solución en función del numero de iteraciones.

[89]:
```
plt.plot(logger.values)
plt.xlabel('Iteración')
plt.ylabel('Valor de la función objetivo')
plt.title('Optimización mediante Differential Evolution')
plt.xlim(0, logger.iteration)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
plt.savefig('Figuras/evol_de.pdf', format='pdf')
# plt.savefig('Figuras/Presentación/evol_de.pdf', format='pdf')
plt.grid(True)
plt.show()
```

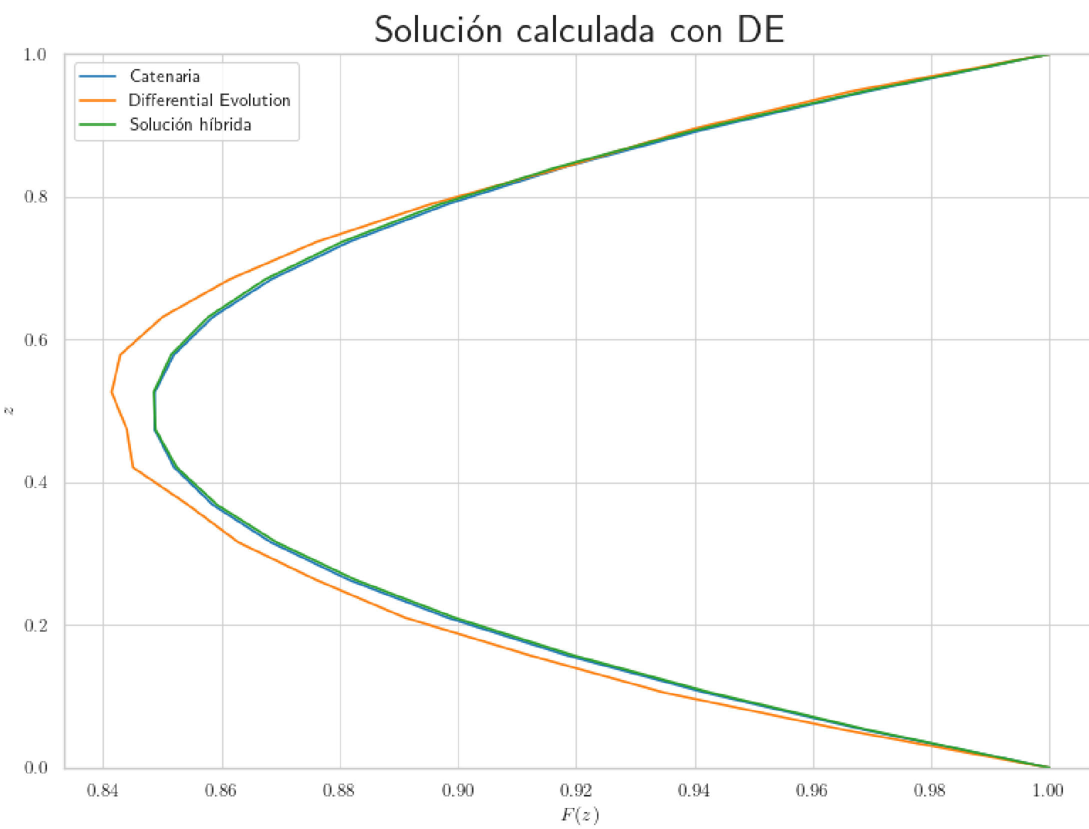


[90]:
```
F_hib = np.concatenate(([F0], sol_hib.x, [F1]))

population_ga = sol_hib.population[1]
F_ga = np.concatenate (([F0], population_ga, [F1]))

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cat = catenaria(z)

error_ga = np.linalg.norm(F_ga - F_cat)/np.linalg.norm(F_cat)
error_hib = np.linalg.norm(F_hib - F_cat)/np.linalg.norm(F_cat)

print('Error con el DE: ', error_ga)
print('Error con el algoritmo híbrido: ', error_hib)
print('Tiempo de ejecución del algoritmo genético: ', time_hib)
```

```
Error con el DE:  0.00609713480067807
Error con el algoritmo híbrido:  0.0009517701018777369
Tiempo de ejecución del algoritmo genético:  448.48453283309937
```

[97]:
```
plt.figure(figsize=(8, 6))
plt.plot(F_cat, z, label='Catenaria')
```

```
plt.plot(F_ga, z, label='Differential Evolution')
plt.plot(F_hib, z, label='Solución híbrida')
plt.legend()
plt.ylabel('$z$')
plt.xlabel('$F(z)$')
plt.title('Solución calculada con DE', fontsize=20)
plt.ylim(0, 1)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.savefig('Figuras/comp_de_cat.pdf', format='pdf')
plt.savefig('Figuras/Presentación/comp_de_cat.pdf', format='pdf',␣
↪bbox_inches='tight', pad_inches=0.1)
plt.show()
```



De esta representación, se puede visualizar claramente el efecto del algoritmo considerado. En primer lugar, se calcula una aproximación de la función con una tolerancia deseada, tras la que se corta la iteración de este. De aquí, se toma el individuo con el mejor *fitness* y se refina la solución con un algoritmo de tipo gradiente.

Este procedimiento sigue siendo más costoso computacionalmente que emplear únicamente el algoritmo tipo gradiente pero, genera una solución de compromiso entre la precisión y rapidez de los algoritmos tipo gradiente y la flexibilidad de los algoritmos heurísticos, consiguiendo así un resultado más óptimo que si se emplease solo uno de los 2 tipos de algoritmos.

1.3.3 Pruebas de casos peculiares

Arrancar el método gradiente en la solución

[160]:
```
n = 20
F_init = np.empty(n-2)
F_init.fill(F_0)
```

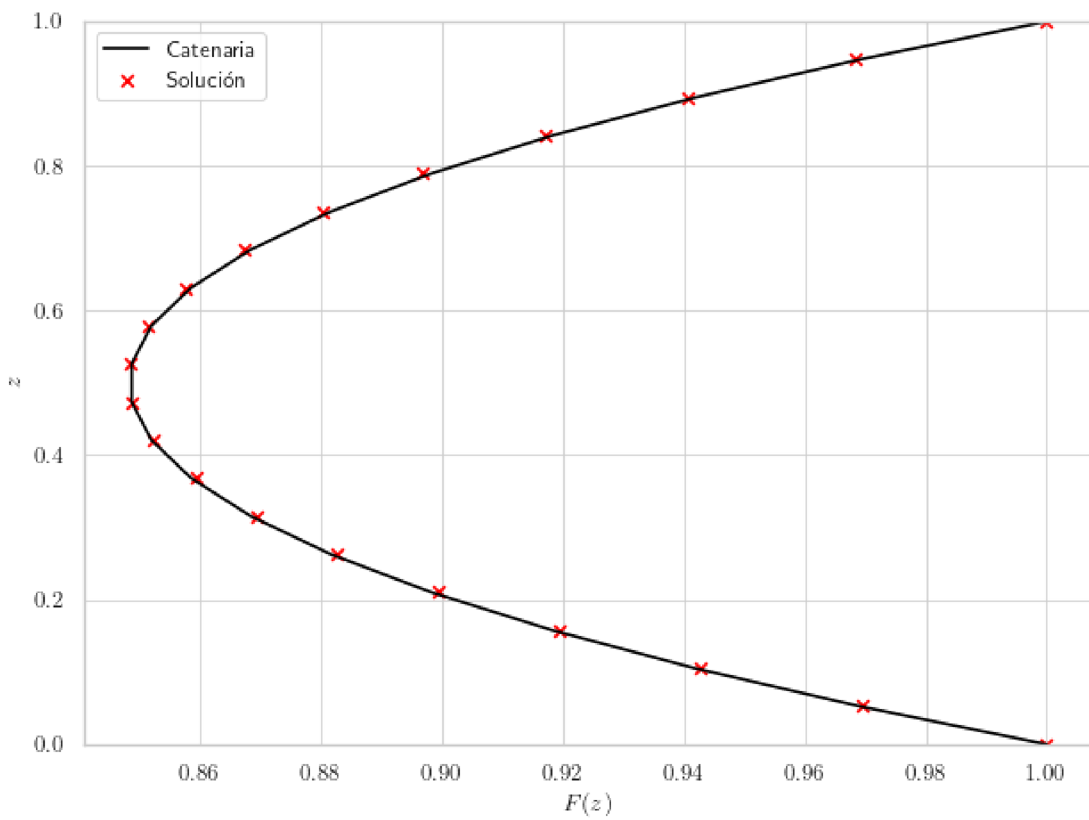
[161]:
```
eps = 0 # soportes iguales
def params_catenaria(params, eps=eps):
    a, k = params
    eq1 = F0 - a * np.cosh(k / a)
    eq2 = F1 - a * np.cosh( (1 + k) / a)
    return [eq1, eq2]

initial_guess = [1.0, 0.0]

a_sol, k_sol = fsolve(params_catenaria, initial_guess)

z = np.linspace(0, 1, n)

def catenaria(z):
    return a_sol * np.cosh((z + k_sol) / a_sol)

F_cat = catenaria(z)
```

[162]:
```
F_init = F_cat[1:-1]

sol_grad = minimize(area_func, F_init, method='SLSQP', tol=1e-16)
F_sol = np.concatenate(([F0], sol_grad.x, [F1]))
error_grad = np.linalg.norm(F_sol - F_cat)/np.linalg.norm(F_cat)
print(sol_grad)
print('Iter:', sol_grad.nit)
print('Norma gradiente: ', np.linalg.norm(sol_grad.jac))
print('Error con el método de gradiente: ', error_grad*100)
```

```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.9537486906223983
       x: [ 9.696e-01  9.428e-01 ...  9.408e-01  9.684e-01]
     nit: 69
     jac: [ 1.192e-07  2.980e-08 ...  1.788e-07  3.725e-08]
    nfev: 1553
    njev: 69
Iter: 69
Norma gradiente:  7.488852788574481e-07
```

Error con el método de gradiente: 0.0950899411560749

[163]:
```
plt.figure()
plt.plot(F_cat, z, label='Catenaria', color='black')
plt.scatter(F_sol, z, label='Solución', marker='x', color='red')
plt.legend()
plt.ylim(0, 1)
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.tight_layout()
plt.savefig('Figuras/sol_F0_especial.pdf', format='pdf')
plt.show()
```



Arrancar el método heurístico en la solución

[195]:
```
F_0 = 1
F0 = F_0
F1 = F_0
n = 20
z = np.linspace(0, 1, n)
F_init = np.empty(n-2)
F_init.fill(F_0)
```

```python
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)

bounds = [(0.5, F_0) for _ in range(n-2)]
```

[196]:
```python
eps = 0 # soportes iguales
def params_catenaria(params, eps=eps):
    a, k = params
    eq1 = F0 - a * np.cosh(k / a)
    eq2 = F1 - a * np.cosh( (1 + k) / a)
    return [eq1, eq2]

initial_guess = [1.0, 0.0]

a_sol, k_sol = fsolve(params_catenaria, initial_guess)

z = np.linspace(0, 1, n)

def catenaria(z):
    return a_sol * np.cosh((z + k_sol) / a_sol)

F_cat = catenaria(z)
```

[197]:
```python
init_pop = np.tile(F_cat[1:-1], (4*n, 1))
# init_pop = np.full((4*n, n-2), F_0)
init_pop.shape
```

[197]:
```
(80, 18)
```

[198]:
```python
sol_hib = differential_evolution(
    area_func,
    bounds=bounds,
    maxiter=100000,
    # popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=1e-16,
    callback=logger,
    init=init_pop,
    # polish = False
)
```

```
Iteración: 1, Valor de la función: 0.9537601425916227
```

[199]:
```python
F_hib = np.concatenate(([F0], sol_hib.x, [F1]))

population_ga = sol_hib.population[1]
```

```
F_ga = np.concatenate (([F0], population_ga, [F1]))

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cat = catenaria(z)

error_ga = np.linalg.norm(F_ga - F_cat)/np.linalg.norm(F_cat)
error_hib = np.linalg.norm(F_hib - F_cat)/np.linalg.norm(F_cat)

print('Error con el DE: ', error_ga)
print('Error con el algoritmo híbrido: ', error_hib)
print('Tiempo de ejecución del algoritmo genético: ', time_hib)
```

```
Error con el DE:  9.767608330210956e-16
Error con el algoritmo híbrido:  0.0009497071847647566
Tiempo de ejecución del algoritmo genético:  0.9514529705047607
```

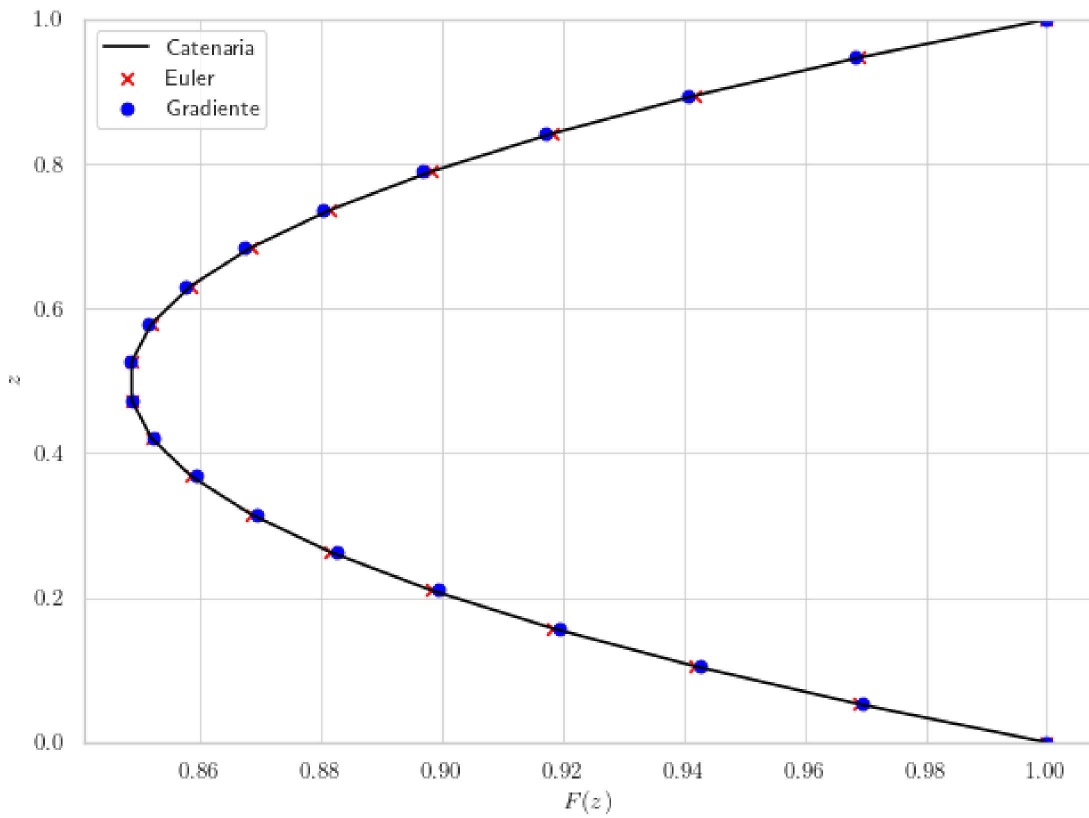
1.3.4 Comparativa con la ecuación de Euler

[203]:
```
n = 20
z = np.linspace(0, 1, n)
F_0 = 1
F0 = F_0
F1 = F_0
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
Fcat = catenaria(z)

F_init = np.empty(n-2)
F_init.fill(F_0)

sol_F0 = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_F0 = np.concatenate(([F0], sol_F0.x, [F1]))
error_F0 = np.linalg.norm(Famp_F0 - Fcat)

error_euler = np.linalg.norm(F_euler - F_cat) / np.linalg.norm(F_cat)
error_euler
```

[203]:
```
2.4178363133322142e-06
```

[211]:
```
plt.show()
plt.plot(Fcat, z, label='Catenaria', color='black')
plt.scatter(F_euler, z, label='Euler', marker='x', color='red')
plt.scatter(F_sol, z, label='Gradiente', marker='o', color='blue')
plt.legend()
```

```
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
# plt.title('Comparación de métodos', fontsize=20)
plt.tight_layout()
plt.savefig('Figuras/sol_euler.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_euler.pdf', format='pdf')
plt.show()
```



1.4 Problema con restricciones de volumen, con soportes iguales.

En el análisis de este apartado se fija el volumen encerrado por la superficie y los planos que contienen a las 2 circunferencias, para ver su efecto sobre la solución del problema.

$V = \int_0^1 F^2 dz$

A continuación, se implementa la expresión siguiente para trasladarla al código de Python.

[215]:
```
def vol_func(F):
    Famp = np.concatenate(([F0], F, [F1]))
    n = np.size(Famp)
    delta_z = 1 / (n - 1)
```

```
    integral = 0
    for i in range(n-1):
        # calcular el valor de la integral
        integrando = Famp[i]**2
        integral += integrando * delta_z

    return (integral - V_lig)
```

1.4.1 Minimzación con un método basado en gradiente

Para poder trabajar comparando las soluciones con las obtenidas para el caso más sencillo, rescatamos los valores de inicialización del problema sin restricciones.

[216]:
```
F_0 = 1
F0 = F_0
F1 = F_0
n = 20
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

z = np.linspace(0, 1, n)

sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_grad = np.concatenate(([F0], sol_grad.x, [F1]))
```

A fin de que los valores no difieran en exceso del valor obtenido del problema sin restricciones, primero se calcula el volumen de la solución con $F_0 = 1$, con la que se partirá para calcular la solución con restricciones.

[217]:
```
V_lig = 0.0 # la ligadura de volumen

vol_no_cons = vol_func(sol_grad.x)
print('Volumen sin restricciones =', vol_no_cons)
```

```
Volumen sin restricciones = 0.8095367935945682
```

Así, se impone la mencionada restricción para observar su influencia en la solución del problema.

[218]:
```
cons = ({'type': 'eq', 'fun': vol_func})
```

[219]:
```
V_lig = 0.6

sol_grad_vol06 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_vol06 = np.concatenate(([F0], sol_grad_vol06.x, [F1]))
```

```python
print(vol_func(Famp_vol06))
```

0.03809525976100969

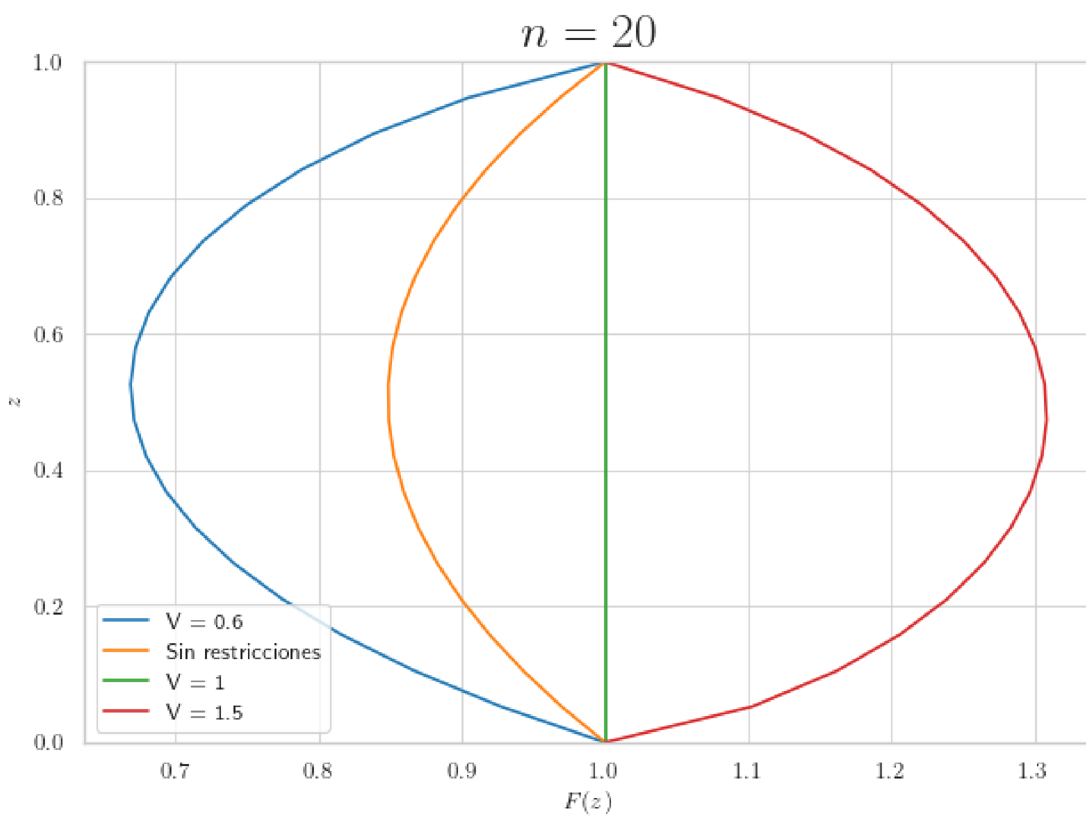
[220]:
```python
V_lig = 1

sol_grad_vol1 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_vol1 = np.concatenate(([F0], sol_grad_vol1.x, [F1]))
vol_func(Famp_vol1)
```

[220]: 4.440892098500626e-16

[221]:
```python
V_lig = 1.5

sol_grad_vol15 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_vol15 = np.concatenate(([F0], sol_grad_vol15.x, [F1]))
vol_func(Famp_vol15)
```

[221]: -0.047618948197209576

[222]:
```python
plt.plot(Famp_vol06, z, label='V = 0.6')
plt.plot(Famp_grad, z, label='Sin restricciones')
plt.plot(Famp_vol1, z, label='V = 1')
plt.plot(Famp_vol15, z, label='V = 1.5')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.legend()
plt.title('$n = {}$'.format(n), fontsize=20)
# plt.savefig('Figuras/vol_inic.pdf', format='pdf')
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
plt.savefig('Figuras/Presentación/vol_inic.pdf', format='pdf')
plt.show()
```



Como cabía de esperar, la forma de la solución se modifica al aumentar el valor de la restricción de volumen. Es importante recalcar que esta restricción no puede tomar cualquier valor, ya que si difiere en gran medida del volumen obtenido sin restricciones, la solución puede resultar inexistente o inestable.

Precisamente, sobre esas posibilidades se va a discutir a continuación.

[223]:
```
F_0 = 1
F0 = F_0
F1 = F_0
n = 100
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

z = np.linspace(0, 1, n)

sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_grad = np.concatenate(([F0], sol_grad.x, [F1]))
```

[224]:
```python
V_lig = 1.6

sol_grad_vol2 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
↪bounds=bounds)
Famp_vol2 = np.concatenate(([F0], sol_grad_vol2.x, [F1]))
```

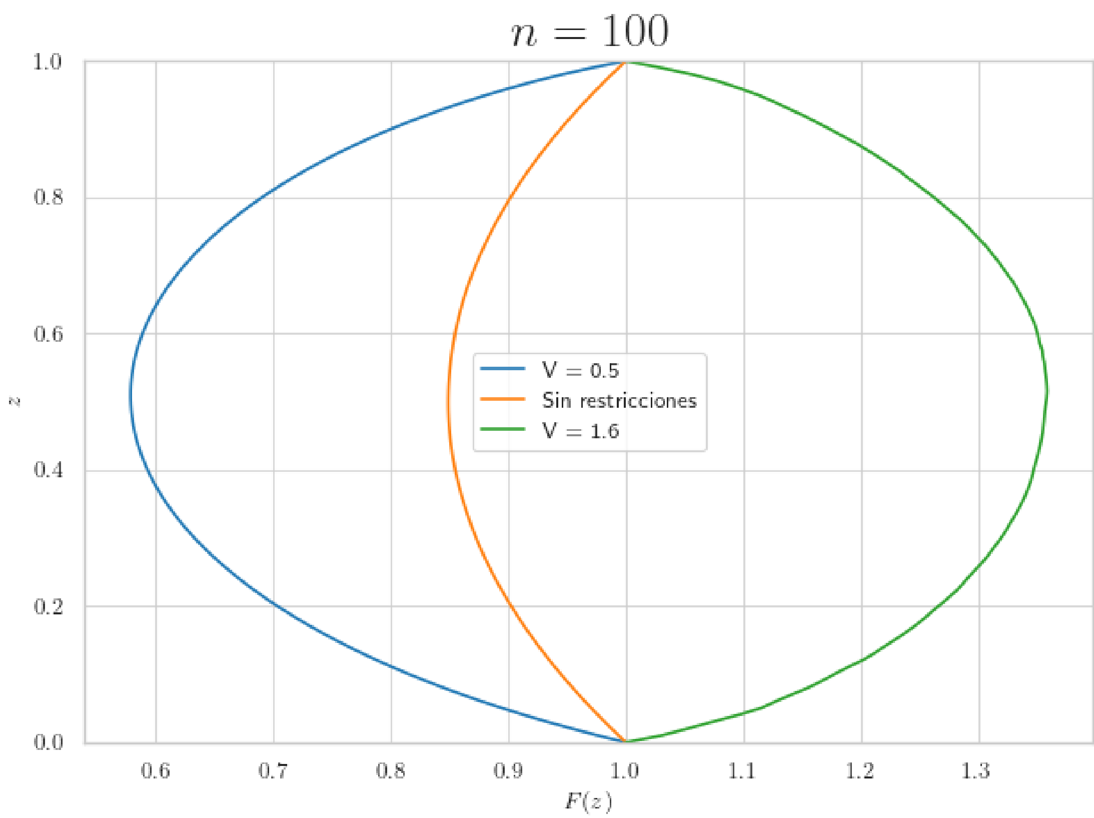
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[224], line 3
      1 V_lig = 1.6
----> 3 sol_grad_vol2 =␣
 ↪minimize(area_func, F_init, constraints=cons, method='SLSQP', bounds=bounds)
      4 Famp_vol2 = np.concatenate(([F0], sol_grad_vol2.x, [F1]))

File c:
 ↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_minimize.
 ↪py:722, in minimize(fun, x0, args, method, jac, hess, hessp, bounds,␣
 ↪constraints, tol, callback, options)
    719     res = _minimize_cobyla(fun, x0, args, constraints, callback=callback,
    720                            bounds=bounds, **options)
    721 elif meth == 'slsqp':
--> 722     res = _minimize_slsqp(fun, x0, args, jac, bounds,
    723                           constraints, callback=callback, **options)
    724 elif meth == 'trust-constr':
    725     res = _minimize_trustregion_constr(fun, x0, args, jac, hess, hessp,
    726                                        bounds, constraints,
    727                                        callback=callback, **options)

File c:
 ↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_slsqp_py.
 ↪py:441, in _minimize_slsqp(func, x0, args, jac, bounds, constraints, maxiter,␣
 ↪ftol, iprint, disp, eps, callback, finite_diff_rel_step, **unknown_options)
    438     c = _eval_constraint(x, cons)
    440 if mode == -1:  # gradient evaluation required
--> 441     g = append(wrapped_grad(x), 0.0)
    442     a = _eval_con_normals(x, cons, la, n, m, meq, mieq)
    444 if majiter > majiter_prev:
    445     # call callback if major iteration has incremented

File c:
 ↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_optimize.
 ↪py:302, in _clip_x_for_func.<locals>.eval(x)
    300 def eval(x):
    301     x = _check_clip_x(x, bounds)
--> 302     return func(x)

File c:
 ↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_differentiable_function
 ↪py:284, in ScalarFunction.grad(self, x)
```

```
    282 if not np.array_equal(x, self.x):
    283     self._update_x_impl(x)
--> 284 self._update_grad()
    285 return self.g

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different able_function
↪py:267, in ScalarFunction._update_grad(self)
    265 def _update_grad(self):
    266     if not self.g_updated:
--> 267         self._update_grad_impl()
    268         self.g_updated = True

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different able_function
↪py:181, in ScalarFunction.__init__.<locals>.update_grad()
    179 self._update_fun()
    180 self.ngev += 1
--> 181 self.g = approx_derivative(fun_wrapped, self.x, f0=self.f,
    182                            **finite_diff_options)

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_numdiff.p :
↪519, in approx_derivative(fun, x0, method, rel_step, abs_step, f0, bounds,␣
↪sparsity, as_linear_operator, args, kwargs)
    516     use_one_sided = False
    518 if sparsity is None:
--> 519     return _dense_difference(fun_wrapped, x0, f0, h,
    520                              use_one_sided, method)
    521 else:
    522     if not issparse(sparsity) and len(sparsity) == 2:

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_numdiff.p :
↪584, in _dense_difference(fun, x0, f0, h, use_one_sided, method)
    582 n = x0.size
    583 J_transposed = np.empty((n, m))
--> 584 h_vecs = np.diag(h)
    586 for i in range(h.size):
    587     if method == '2-point':

File c:\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\numpy\lib\twodim_base
↪py:293, in diag(v, k)
    291 if len(s) == 1:
    292     n = s[0]+abs(k)
--> 293     res = zeros((n, n), v.dtype)
    294     if k >= 0:
    295         i = k
```

```
KeyboardInterrupt:
```

[ ]:
```
V_lig = 0.5

sol_grad_vol05 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
↪bounds=bounds)
Famp_vol05 = np.concatenate(([F0], sol_grad_vol05.x, [F1]))
```

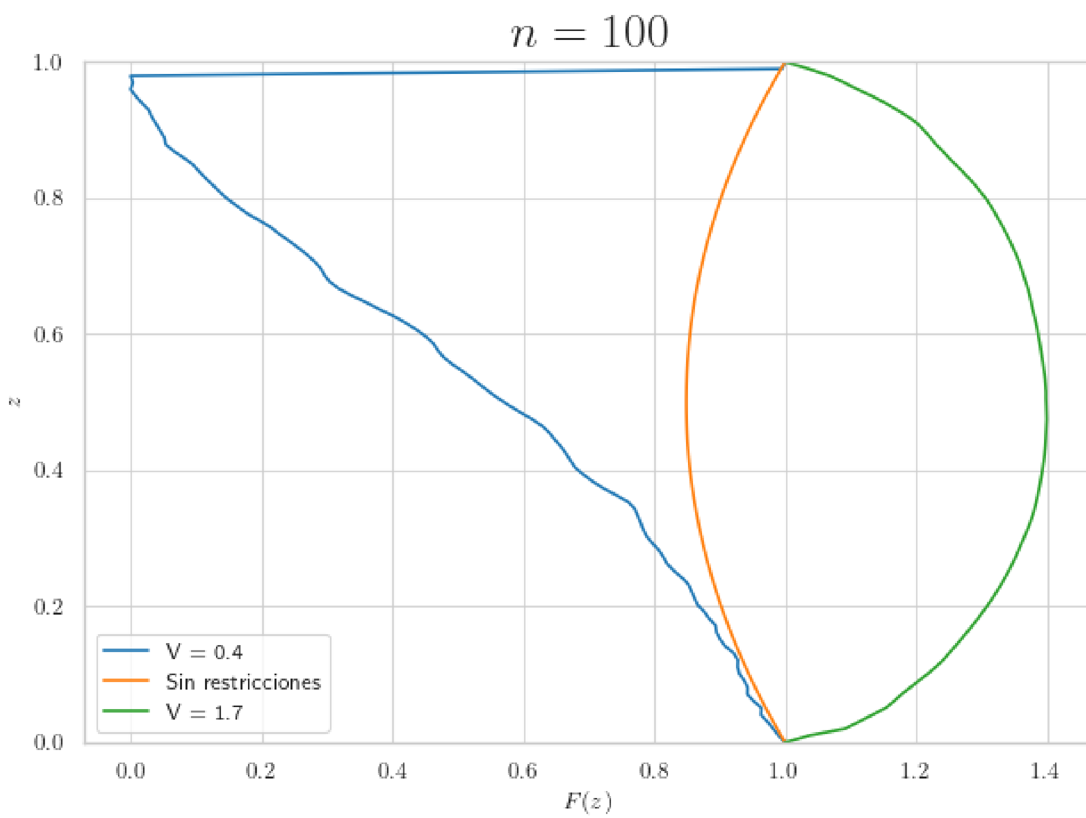
[ ]:
```
plt.plot(Famp_vol05, z, label='V = 0.5')
plt.plot(Famp_grad, z, label='Sin restricciones')
plt.plot(Famp_vol2, z, label='V = 1.6')
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.legend()
plt.title('$n = {}$'.format(n), fontsize=20)
# plt.savefig('Figuras/vol_sol_n100.pdf', format='pdf')
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
plt.savefig('Figuras/Presentación/vol_sol_n100.pdf', format='pdf')
plt.show()
```



```
[ ]: V_lig = 1.7

     sol_grad_vol2 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_vol2 = np.concatenate(([F0], sol_grad_vol2.x, [F1]))
```

```
[ ]: V_lig = 0.4

     sol_grad_vol05 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_vol05 = np.concatenate(([F0], sol_grad_vol05.x, [F1]))
```

```
[ ]: plt.plot(Famp_vol05, z, label='V = 0.4')
     plt.plot(Famp_grad, z, label='Sin restricciones')
     plt.plot(Famp_vol2, z, label='V = 1.7')
     plt.xlabel('$F(z)$')
     plt.ylabel('$z$')
     plt.ylim(0, 1)
     plt.legend()
     plt.title('$n = {}$'.format(n), fontsize=20)
     # plt.savefig('Figuras/vol_no_sol_n100.pdf', format='pdf')
     plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
     plt.savefig('Figuras/Presentación/vol_no_sol_n100.pdf', format='pdf')
     plt.show()
```



Se aprecia que, para valores que se alejan en gran medida de la solución sin restricciones, se generan soluciones que, como poco, no tienen sentido físico.

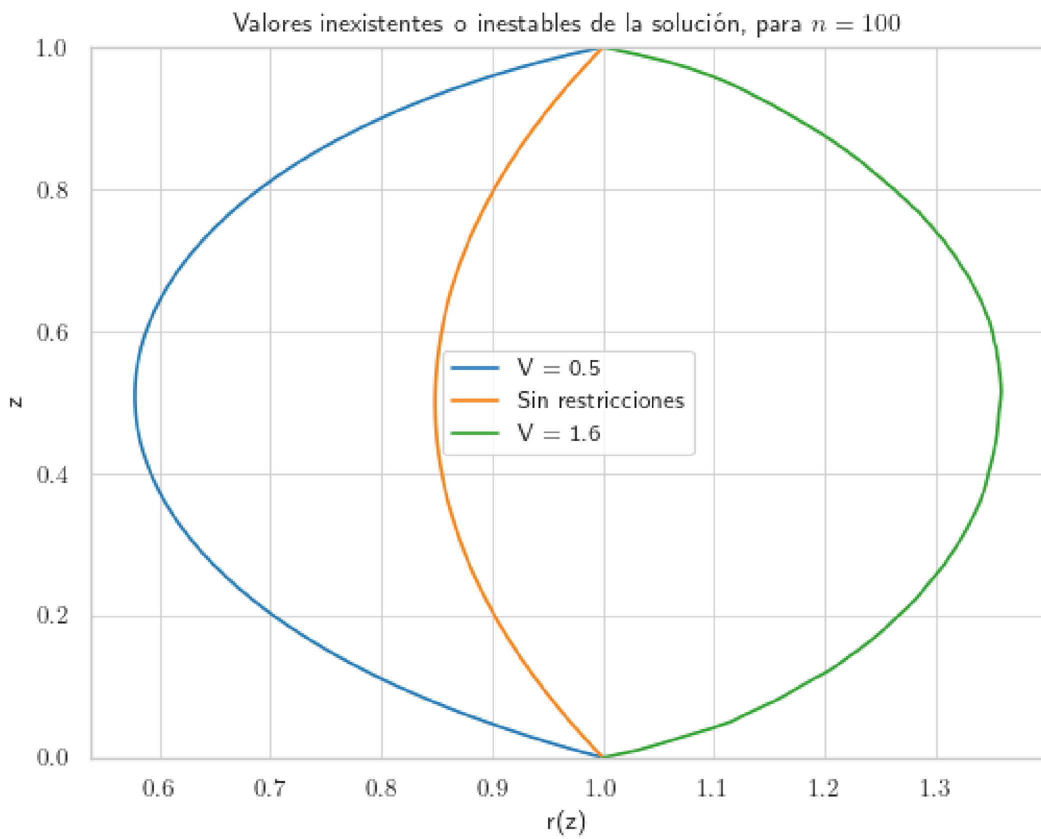
Cabe pensar que, para los casos más sensibles, pueda existir solución solo que el método no tiene la precisión adecuada para calcularlo. Así, se plantea evaluar el efecto de aumentar el número de puntos, a fin de tener una representación más precisa, caso que se expone a continuación.

[ ]:
```
F_0 = 1
F0 = F_0
F1 = F_0
n = 100
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

z = np.linspace(0, 1, n)

sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_grad = np.concatenate(([F0], sol_grad.x, [F1]))
```

```
[ ]: V_lig = 1.6

     sol_grad_vol2 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_vol2 = np.concatenate(([F0], sol_grad_vol2.x, [F1]))
```

```
[ ]: V_lig = 0.5

     sol_grad_vol04 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_vol04 = np.concatenate(([F0], sol_grad_vol04.x, [F1]))
```

```
[ ]: plt.plot(Famp_vol04, z, label='V = 0.5')
     plt.plot(Famp_grad, z, label='Sin restricciones')
     plt.plot(Famp_vol2, z, label='V = 1.6')
     plt.legend()
     plt.xlabel('r(z)')
     plt.ylabel('z')
     plt.ylim(0, 1)
     plt.title('Valores inexistentes o inestables de la solución, para $n = {}$'.
      ↪format(n))
     plt.grid(True)
     plt.show()
```



Tras analizar el resultado de aumentar el número de puntos, se concluye que, efectivamente, para dichos valores de la restricción de volumen, el problema directamente no tiene solución.

Influencia de la distribución de puntos. En este caso, se va a considerar el efecto de considerar diferentes distribuciones de puntos en las que dividir el intervalo del problema.

Una vez más, se consideran el resto de parámetros idénticos al primer caso de estudio, salvo por el número de puntos n, que se va a modificar a fin de apreciar el comportamiento de distintas distribuciones de puntos en los casos más exigentes, es decir, en los casos extremos de n muy grande o muy pequeño. El caso de n intermedio se omite por presentar soluciones casi idénticas en todos los casos.

Caso de n muy pequeño. En primer lugar, se considera la restricción de volumen que generaría un cilindro recto. En este caso, que se traduce en $V = 1.0$.

[ ]:
```
n = 20
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

```
V_lig = 1.0

# planteamos la función que calcula el área tomando delta_z como un array␣
↪definido anteriormente
def area_distrib(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)

    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z[i] # esquema adelantado para el␣
↪cálculo de la derivada
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z[i]

    return integral
```

[ ]:
```
def vol_distrib(F):
    Famp = np.concatenate(([F0], F, [F1]))
    n = np.size(Famp)
    # delta_z = 1 / (n - 1)

    integral = 0
    for i in range(n-1):
        # calcular el valor de la integral
        integrando = Famp[i]**2
        integral += integrando * delta_z[i]

    return (integral - V_lig)

# V_lig = 1.0
cons = ({'type': 'eq', 'fun': vol_distrib})
```

- Distribución de puntos **equiespaciada**

[ ]:
```
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
↪bounds=bounds)
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
print('Volumen de la solución equiespaciada:', vol_distrib(sol_gradeq.x) + V_lig)
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_cateq = catenaria(zeq)
# error_eq = np.linalg.norm(Famp_eq - F_cateq)/np.linalg.norm(F_cateq)
# print('El error para distribución equiespaciada es:', error_eq)
```

```
Volumen de la solución equiespaciada: 1.0
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**: $z_i = \cos\left(\frac{\frac{\pi}{2}+\pi i}{n+1}\right), \quad i = 0, \ldots, n$

```
[ ]: def chebyshev_zeros(a, b, n):
         cheb_zeros = np.cos((2 * np.arange(1, n+1) - 1) * np.pi / (2 * n))
         mapped_zeros = 0.5 * (b - a) * (cheb_zeros + 1) + a
         sorted_zeros = np.sort(mapped_zeros) # porque los calcula en orden␣
      ↪decreciente
         return sorted_zeros

     zchebz = chebyshev_zeros(0, 1, n)
     delta_z = np.diff(zchebz)
     sol_gradchebz = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
     print('Volumen de la solución con ceros de Chebyshev:',␣
      ↪vol_distrib(sol_gradchebz.x) + V_lig)
     # a_sol, k_sol = fsolve(params_catenaria, initial_guess)
     # F_catchebz = catenaria(zchebz)
     # error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)/np.linalg.
      ↪norm(F_catchebz)
     # print('El error para los nodos de Chebyshev es:', error_chebz)
```

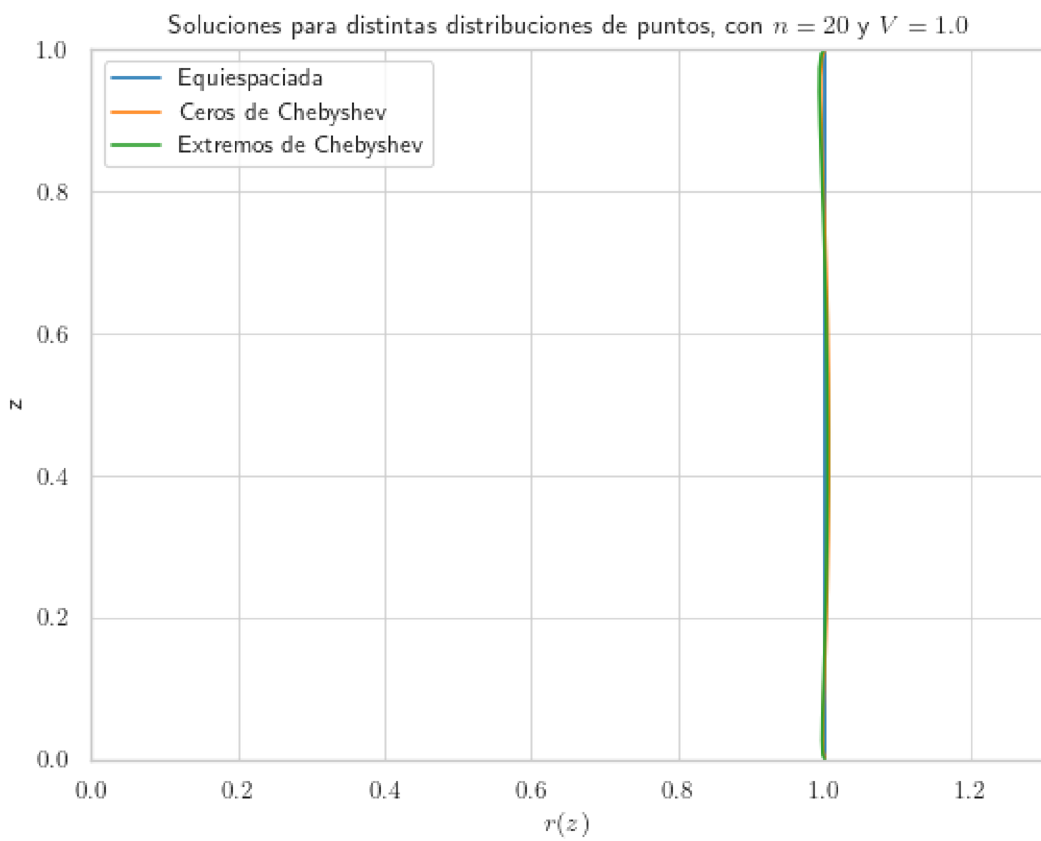
```
Volumen de la solución con ceros de Chebyshev: 1.0000001157114373
```

- Distribución de puntos según un esquema de **extremos de Chebyshev**: $z_i = \cos\left(\frac{\pi i}{n}\right), \quad i = 0, \ldots, n$

```
[ ]: def chebyshev_extremes(a, b, n):
         cheb_extremes = np.cos(np.arange(n) * np.pi / (n - 1))
         mapped_extremes = 0.5 * (b - a) * (cheb_extremes + 1) + a
         sorted_extremes = np.sort(mapped_extremes)
         return sorted_extremes

     zchebe = chebyshev_extremes(0, 1, n)
     delta_z = np.diff(zchebe)
     sol_gradchebe = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
     print('Volumen de la solución con extremos de Chebyshev:',␣
      ↪vol_distrib(sol_gradchebe.x) + V_lig)
     # a_sol, k_sol = fsolve(params_catenaria, initial_guess)
     # F_catchebe = catenaria(zchebe)
     # error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)/np.linalg.
      ↪norm(F_catchebe)
     # print('El error para los extremos de Chebyshev es:', error_chebe)
```

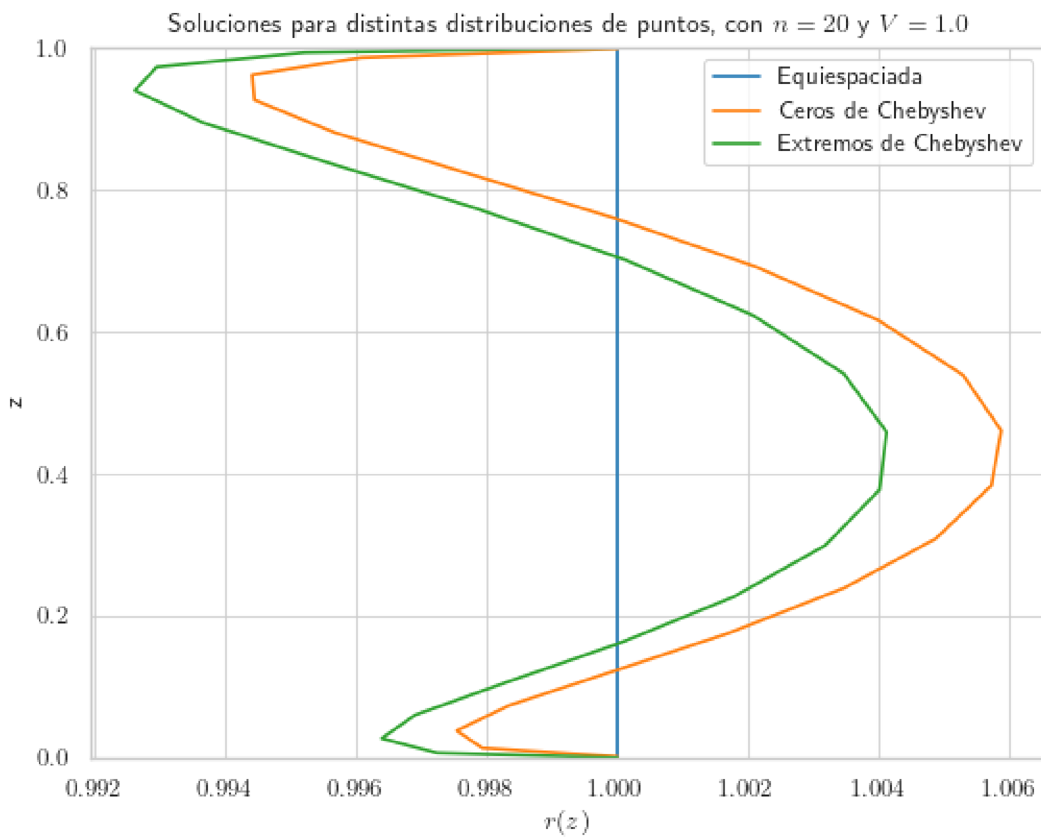
Volumen de la solución con extremos de Chebyshev: 1.0000000024621918

[ ]:
```
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
# plt.plot(F_cateq, zeq, label='Catenaria')
plt.ylabel('z')
plt.xlabel('$r(z)$')
plt.ylim(0, 1)
plt.xlim(0, 1.3)
plt.title('Soluciones para distintas distribuciones de puntos, con $n = {}$ y $V␣
↪= {}$'.format(n, V_lig))
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
# plt.plot(F_cateq, zeq, label='Catenaria')
```

```
plt.ylabel('z')
plt.xlabel('$r(z)$')
plt.ylim(0, 1)
plt.title('Soluciones para distintas distribuciones de puntos, con $n = {}$ y $V␣
↪= {}$'.format(n, V_lig))
plt.legend()
plt.grid(True)
plt.show()
```



A primera vista, todas las soluciones son extremadamente similares, por lo que se pueden considerar todas igual de válidas. Sin embargo, al ampliar y observar la solución de una forma más cercana, se aprecia que la única que cumple la restricción de $V = 1$ y además se corresponde con la solución esperada es la equiespaciada.

NO SÉ EXPLICAR POR QUÉ ESO ES ASÍ

Por otro lado, ahora se va a considerar la restricción de volumen de $V = 1.3$, es decir, que la burbuja sobresalga y el volumen sea mayor que el correspondiente a un cilindro recto.

[ ]:
```
n = 20
F_init = np.empty(n-2)
```

```
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
V_lig = 1.3

# planteamos la función que calcula el área tomando delta_z como un array␣
↪definido anteriormente
def area_distrib(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)

    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z[i] # esquema adelantado para el␣
↪cálculo de la derivada
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z[i]

    return integral
```

[ ]:
```
def vol_distrib(F):
    Famp = np.concatenate(([F0], F, [F1]))
    n = np.size(Famp)
    # delta_z = 1 / (n - 1)

    integral = 0
    for i in range(n-1):
        # calcular el valor de la integral
        integrando = Famp[i]**2
        integral += integrando * delta_z[i]

    return (integral - V_lig)

# V_lig = 1.0
cons = ({'type': 'eq', 'fun': vol_distrib})
```

- Distribución de puntos **equiespaciada**

[ ]:
```
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
↪bounds=bounds)
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
print('Volumen de la solución equiespaciada:', vol_distrib(sol_gradeq.x) + V_lig)
```

```
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_cateq = catenaria(zeq)
# error_eq = np.linalg.norm(Famp_eq - F_cateq)/np.linalg.norm(F_cateq)
# print('El error para distribución equiespaciada es:', error_eq)
```

```
Volumen de la solución equiespaciada: 1.300000020521231
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**: $z_i = \cos\left(\frac{\frac{\pi}{2}+\pi i}{n+1}\right), \quad i = 0, \ldots, n$

```
[ ]: def chebyshev_zeros(a, b, n):
         cheb_zeros = np.cos((2 * np.arange(1, n+1) - 1) * np.pi / (2 * n))
         mapped_zeros = 0.5 * (b - a) * (cheb_zeros + 1) + a
         sorted_zeros = np.sort(mapped_zeros) # porque los calcula en orden␣
      ↪decreciente
         return sorted_zeros

     zchebz = chebyshev_zeros(0, 1, n)
     delta_z = np.diff(zchebz)
     sol_gradchebz = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
     print('Volumen de la solución con ceros de Chebyshev:',␣
      ↪vol_distrib(sol_gradchebz.x) + V_lig)
     # a_sol, k_sol = fsolve(params_catenaria, initial_guess)
     # F_catchebz = catenaria(zchebz)
     # error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)/np.linalg.
      ↪norm(F_catchebz)
     # print('El error para los nodos de Chebyshev es:', error_chebz)
```

```
Volumen de la solución con ceros de Chebyshev: 1.3000001343495755
```

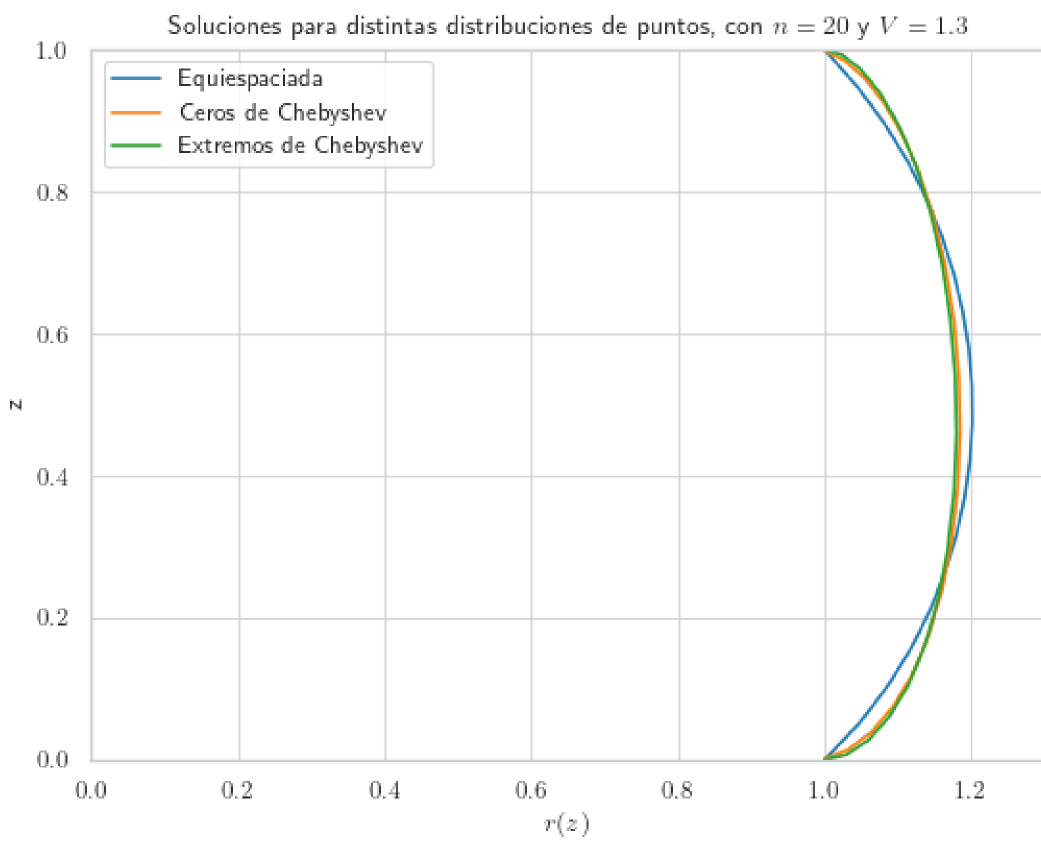
- Distribución de puntos según un esquema de **extremos de Chebyshev**: $z_i = \cos\left(\frac{\pi i}{n}\right), \quad i = 0, \ldots, n$

```
[ ]: def chebyshev_extremes(a, b, n):
         cheb_extremes = np.cos(np.arange(n) * np.pi / (n - 1))
         mapped_extremes = 0.5 * (b - a) * (cheb_extremes + 1) + a
         sorted_extremes = np.sort(mapped_extremes)
         return sorted_extremes

     zchebe = chebyshev_extremes(0, 1, n)
     delta_z = np.diff(zchebe)
     sol_gradchebe = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
      ↪bounds=bounds)
     Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
     print('Volumen de la solución con extremos de Chebyshev:',␣
      ↪vol_distrib(sol_gradchebe.x) + V_lig)
```

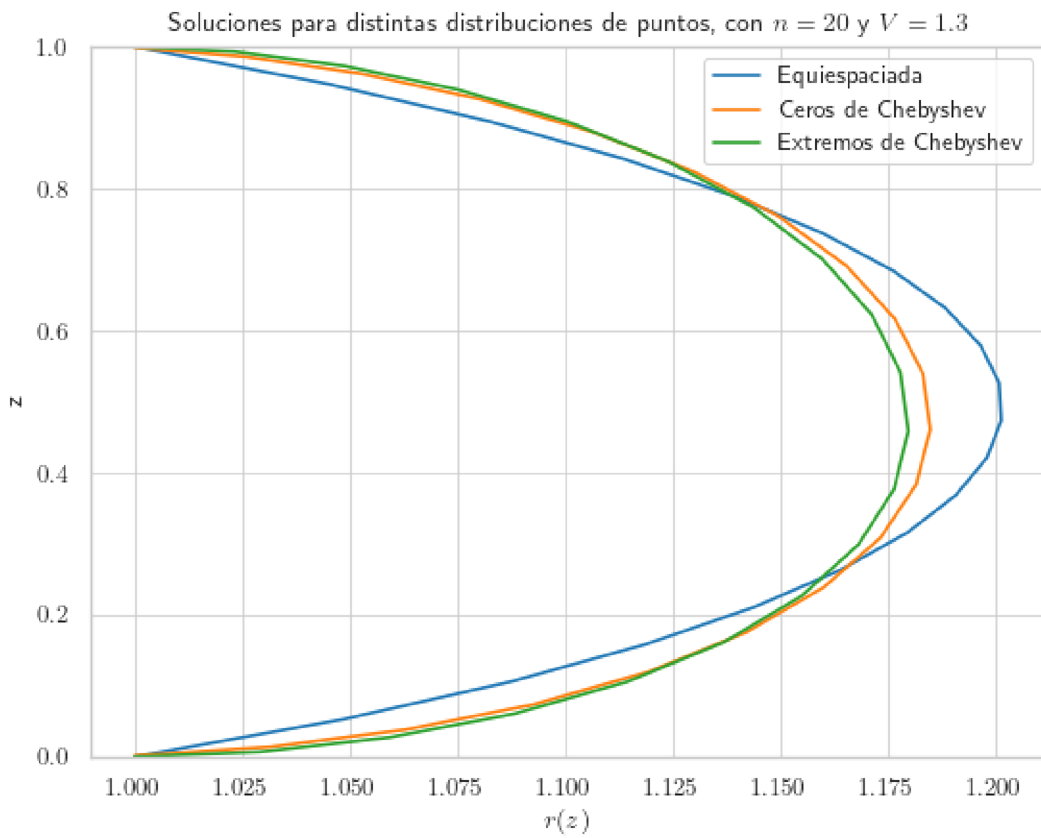
```
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_catchebe = catenaria(zchebe)
# error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)/np.linalg.
↪norm(F_catchebe)
# print('El error para los extremos de Chebyshev es:', error_chebe)
```

Volumen de la solución con extremos de Chebyshev: 1.3000002438497287

[ ]:
```
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
# plt.plot(F_cateq, zeq, label='Catenaria')
plt.ylabel('z')
plt.xlabel('$r(z)$')
plt.ylim(0, 1)
plt.xlim(0, 1.3)
plt.title('Soluciones para distintas distribuciones de puntos, con $n = {}$ y $V␣
↪= {}$'.format(n, V_lig))
plt.legend()
plt.grid(True)
plt.show()
```



```
[ ]: plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
     plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
     plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
     # plt.plot(F_cateq, zeq, label='Catenaria')
     plt.ylabel('z')
     plt.xlabel('$r(z)$')
     plt.ylim(0, 1)
     plt.title('Soluciones para distintas distribuciones de puntos, con $n = {}$ y $V␣
      ↪= {}$'.format(n, V_lig))
     plt.legend()
     plt.grid(True)
     plt.show()
```



Para este caso, también se observa que se tienen soluciones muy dispares en todos los métodos considerados. Salvo la producida por la distribución que concentra más puntos en el centro, todas parecen plausibles y es difícil discernir cual de ellas es la solución adecuada.

Caso de n grande.

[ ]:
```
n = 100
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
V_lig = 1.0

# planteamos la función que calcula el área tomando delta_z como un array␣
↪definido anteriormente
def area_distrib(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)
    # delta_z = 1 / (n - 1)

    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z[i] # esquema adelantado para el␣
↪cálculo de la derivada
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z[i]

    return integral
```

- Distribución de puntos **equiespaciada**

[ ]:
```
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
↪bounds=bounds)
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
print('Volumen de la solución equiespaciada:', vol_distrib(sol_gradeq.x) + V_lig)
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_cateq = catenaria(zeq)
# error_eq = np.linalg.norm(Famp_eq - F_cateq)
# print('El error para distribución equiespaciada es:', error_eq)
```

```
Volumen de la solución equiespaciada: 1.0
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**

[ ]:
```
def chebyshev_zeros(a, b, n):
    cheb_zeros = np.cos((2 * np.arange(1, n+1) - 1) * np.pi / (2 * n))
    mapped_zeros = 0.5 * (b - a) * (cheb_zeros + 1) + a
    sorted_zeros = np.sort(mapped_zeros) # porque los calcula en orden␣
↪decreciente
```

```
    return sorted_zeros

zchebz = chebyshev_zeros(0, 1, n)
delta_z = np.diff(zchebz)
sol_gradchebz = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
print('Volumen de la solución con ceros de Chebyshev:',␣
 ↪vol_distrib(sol_gradchebz.x) + V_lig)
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_catchebz = catenaria(zchebz)
# error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)
# print('El error para los nodos de Chebyshev es:', error_chebz)
```

```
Volumen de la solución con ceros de Chebyshev: 0.999981358695781
```

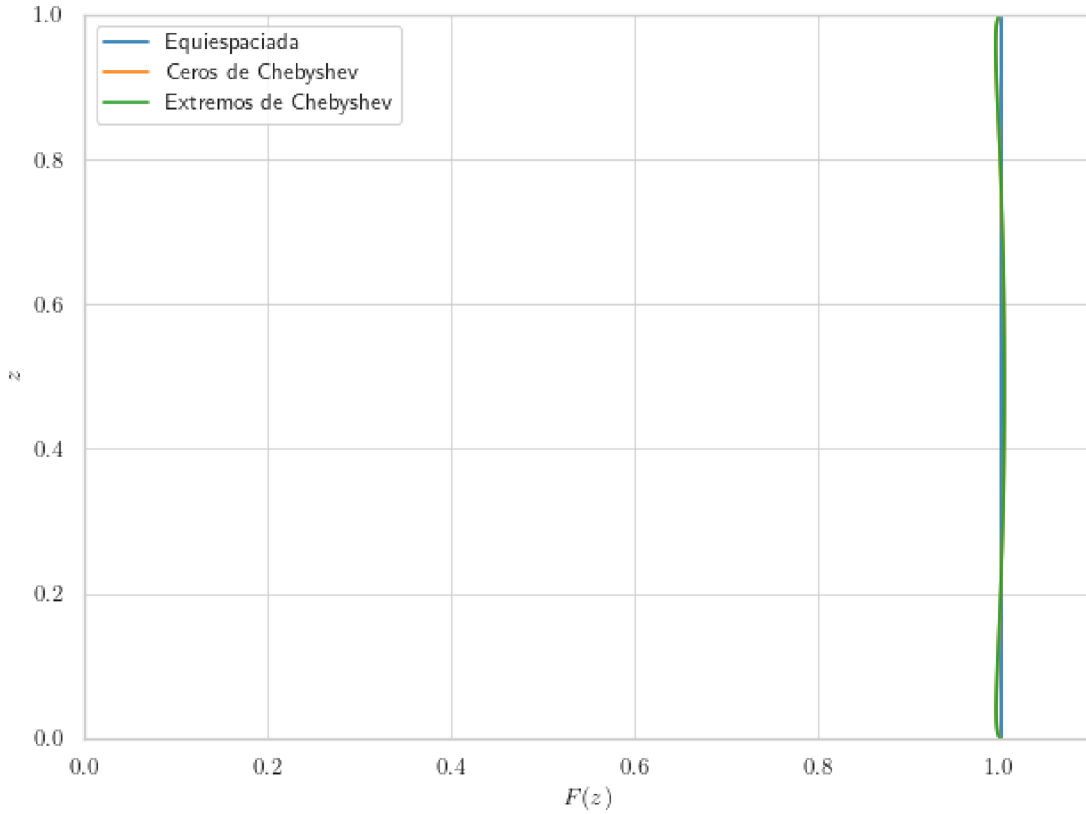
- Distribución de puntos según un esquema de **extremos de Chebyshev**

[ ]:
```
def chebyshev_extremes(a, b, n):
    cheb_extremes = np.cos(np.arange(n) * np.pi / (n - 1))
    mapped_extremes = 0.5 * (b - a) * (cheb_extremes + 1) + a
    sorted_extremes = np.sort(mapped_extremes)
    return sorted_extremes

zchebe = chebyshev_extremes(0, 1, n)
delta_z = np.diff(zchebe)
sol_gradchebe = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
print('Volumen de la solución con extremos de Chebyshev:',␣
 ↪vol_distrib(sol_gradchebe.x) + V_lig)
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_catchebe = catenaria(zchebe)
# error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)
# print('El error para los extremos de Chebyshev es:', error_chebe)
```

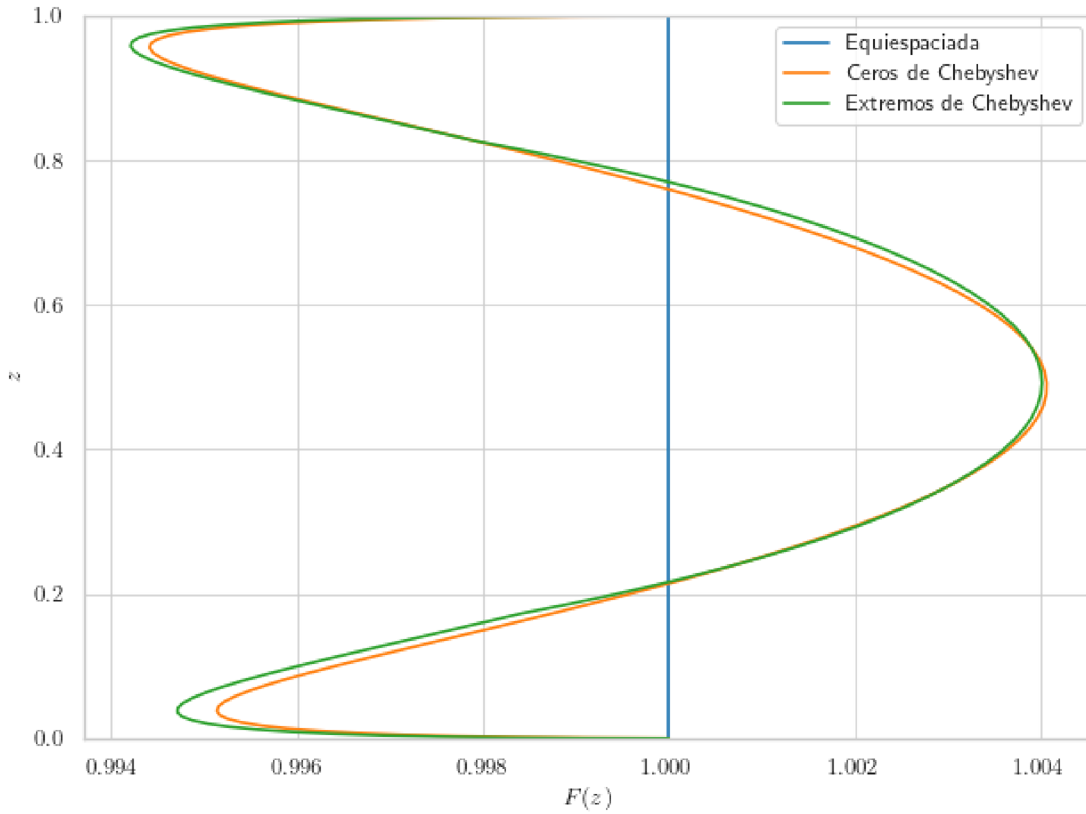
```
Volumen de la solución con extremos de Chebyshev: 1.000000639539971
```

[ ]:
```
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.xlim(0, 1.1)
plt.legend()
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
```

```
# plt.title('Soluciones para distintas distribuciones de puntos, con $n = {}$ y␣
↪$V = {}$'.format(n, V_lig))
plt.savefig('Figuras/sol_vol1_lejos.pdf', format='pdf')
plt.show()
```




```
[ ]: plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
# plt.xlim(0, 1.1)
plt.legend()
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.title('Soluciones para distintas distribuciones de puntos, con $n = {}$ y␣
↪$V = {}$'.format(n, V_lig))
plt.savefig('Figuras/sol_vol1_cerca.pdf', format='pdf')
plt.show()
```



Aquí, una vez más, las 4 soluciones son esencialmente idénticas, pero, al aumentar la resolución, se aprecia claramente el mismo fenómeno que se observaba con pocos puntos.

Una vez más, se va a considerar el problema con una condición de volumen superior a la del cilindro recto, para observar como se comporta la solución en el caso que se consideran tantos puntos.

[ ]:
```
n = 100
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
V_lig = 1.3
```

- Distribución de puntos **equiespaciada**

[ ]:
```
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
print('Volumen de la solución equiespaciada:', vol_distrib(sol_gradeq.x) + V_lig)
```

```
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_cateq = catenaria(zeq)
# error_eq = np.linalg.norm(Famp_eq - F_cateq)/np.linalg.norm(F_cateq)
# print('El error para distribución equiespaciada es:', error_eq)
```

```
Volumen de la solución equiespaciada: 1.3019670464246735
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**: $z_i = \cos\left(\frac{\frac{\pi}{2}+\pi i}{n+1}\right), \quad i = 0, \ldots, n$

[ ]:
```
def chebyshev_zeros(a, b, n):
    cheb_zeros = np.cos((2 * np.arange(1, n+1) - 1) * np.pi / (2 * n))
    mapped_zeros = 0.5 * (b - a) * (cheb_zeros + 1) + a
    sorted_zeros = np.sort(mapped_zeros) # porque los calcula en orden␣
 ↪decreciente
    return sorted_zeros

zchebz = chebyshev_zeros(0, 1, n)
delta_z = np.diff(zchebz)
sol_gradchebz = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
print('Volumen de la solución con ceros de Chebyshev:',␣
 ↪vol_distrib(sol_gradchebz.x) + V_lig)
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_catchebz = catenaria(zchebz)
# error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)/np.linalg.
 ↪norm(F_catchebz)
# print('El error para los nodos de Chebyshev es:', error_chebz)
```

```
Volumen de la solución con ceros de Chebyshev: 1.3027974843705101
```

- Distribución de puntos según un esquema de **extremos de Chebyshev**: $z_i = \cos\left(\frac{\pi i}{n}\right), \quad i = 0, \ldots, n$

[ ]:
```
def chebyshev_extremes(a, b, n):
    cheb_extremes = np.cos(np.arange(n) * np.pi / (n - 1))
    mapped_extremes = 0.5 * (b - a) * (cheb_extremes + 1) + a
    sorted_extremes = np.sort(mapped_extremes)
    return sorted_extremes

zchebe = chebyshev_extremes(0, 1, n)
delta_z = np.diff(zchebe)
sol_gradchebe = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
print('Volumen de la solución con extremos de Chebyshev:',␣
 ↪vol_distrib(sol_gradchebe.x) + V_lig)
```

```
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_catchebe = catenaria(zchebe)
# error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)/np.linalg.
 ↪norm(F_catchebe)
# print('El error para los extremos de Chebyshev es:', error_chebe)
```

```
Volumen de la solución con extremos de Chebyshev: 1.3029239996571174
```

- Distribución que **concentra puntos en el centro del intervalo**.

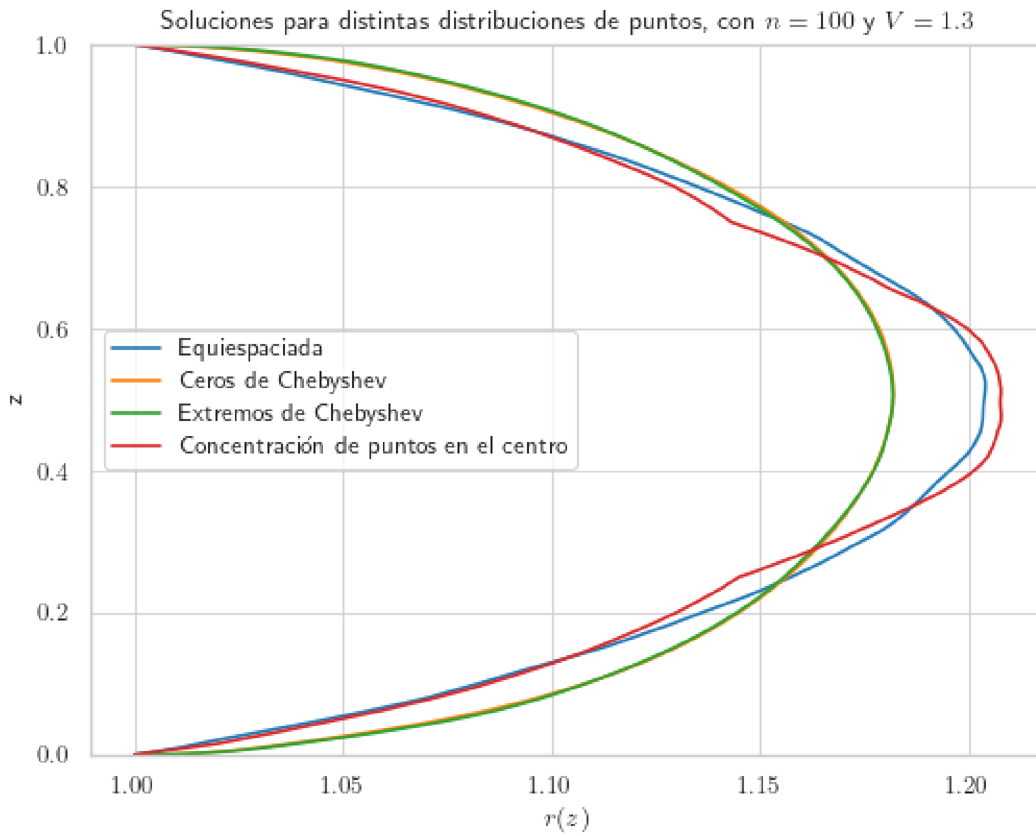
Simplemente en la mitad central del intervalo se concentran puntos a la mitad de distancia que en los extremos.

[ ]:
```
def concentrar_centro(a, b, n):
  n1 = n // 6
  n2 = n - 2 * n1
  n3 = n - n1 - n2

  x1 = np.linspace(a, a + (b - a) / 4, n1, endpoint=False)
  x2 = np.linspace(a + (b - a) / 4, a + 3 * (b - a) / 4, n2, endpoint=False)
  x3 = np.linspace(a + 3 * (b - a) / 4, b, n3)

  x = np.concatenate((x1, x2, x3))
  return x

zcent = concentrar_centro(0, 1, n)
delta_z = np.diff(zcent)
sol_gradcent = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_cent = np.concatenate(([F0], sol_gradcent.x, [F1]))
print('Volumen de la solución concentrada en el centro:',␣
 ↪vol_distrib(sol_gradcent.x) + V_lig)
# a_sol, k_sol = fsolve(params_catenaria, initial_guess)
# F_catcent = catenaria(zcent)
# error_cent = np.linalg.norm(Famp_cent - F_catcent)/np.linalg.norm(F_catcent)
# print('El error para la distribución de puntos concentrada en el centro es:',␣
 ↪error_cent)
```

```
Volumen de la solución concentrada en el centro: 1.3000017620008701
```

[ ]:
```
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
plt.plot(Famp_cent, zcent, label='Concentración de puntos en el centro')
# plt.plot(F_cateq, zeq, label='Catenaria')
plt.ylabel('z')
plt.xlabel('$r(z)$')
plt.ylim(0, 1)
```

```
plt.title('Soluciones para distintas distribuciones de puntos, con $n = {}$ y $V␣
↪= {}$'.format(n, V_lig))
plt.legend()
plt.grid(True)
plt.show()
```



Para este caso, se observa que ninguna de los esquemas empleados proporciona una solución convincente. Por esto, se puede deducir que, la introducción de complicaciones en el cálculo de la solución (como pueden ser las restricciones), junto al cálculo de la solución para una cantidad importante de puntos, como se observa en el apartado anterior, dificulta mucho el proceso de optimización, denotando la necesidad de métodos más sofisticados que se escapan de el análisis de este trabajo.

1.4.2 Minimización con un método heurístico (Differential evolution)

[ ]:
```
F_0 = 1
n = 20
z = np.linspace(0, 1, n)

bounds = [(F_0-0.5, F_0) for _ in range(n-2)]
```

```
tol_vol = 1e-2 # tolerancia con la restricción
cons_de = NonlinearConstraint(vol_func, -tol_vol, tol_vol)
```

[ ]:
```
V_lig = 0
vol_de = differential_evolution(
    area_func,
    bounds=bounds,
    # constraints=cons_de,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=0.01,
    # disp=True,
    callback=logger
)

Famp_vol_de = np.concatenate(([F0], vol_de.x, [F1]))


# Graficar los resultados
plt.plot(logger.values)
plt.xlabel('Iteración')
plt.ylabel('Valor de la función objetivo')
plt.title('Optimización Diferencial Evolutiva')
plt.show()
```

```
Iteración: 1, Valor de la función: 1.477716569978038
Iteración: 2, Valor de la función: 1.309425926962669
Iteración: 3, Valor de la función: 1.2763531152768661
Iteración: 4, Valor de la función: 1.2763531152768661
Iteración: 5, Valor de la función: 1.2763531152768661
Iteración: 6, Valor de la función: 1.2763531152768661
Iteración: 7, Valor de la función: 1.2763531152768661
Iteración: 8, Valor de la función: 1.272790913776141
Iteración: 9, Valor de la función: 1.22558523906353
Iteración: 10, Valor de la función: 1.22558523906353
Iteración: 11, Valor de la función: 1.2044619579912867
Iteración: 12, Valor de la función: 1.1922202725517823
Iteración: 13, Valor de la función: 1.1428455733686522
Iteración: 14, Valor de la función: 1.1428455733686522
Iteración: 15, Valor de la función: 1.1428455733686522
Iteración: 16, Valor de la función: 1.1134951695027295
Iteración: 17, Valor de la función: 1.1134951695027295
Iteración: 18, Valor de la función: 1.1134951695027295
Iteración: 19, Valor de la función: 1.1134951695027295
Iteración: 20, Valor de la función: 1.1134951695027295
```

```
Iteración: 21, Valor de la función: 1.1129220880703399
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[819], line 2
      1 V_lig = 0
----> 2 vol_de = differential_evolution(
      3     area_func,
      4     bounds=bounds,
      5     # constraints=cons_de,
      6     maxiter=100000,
      7     popsize=4*n,
      8     mutation=(0.5, 1),
      9     recombination=0.7,
     10     strategy='best1bin',
     11     tol=0.01,
     12     # disp=True,
     13     callback=logger
     14 )
     16 Famp_vol_de = np.concatenate(([F0], vol_de.x, [F1]))
     19 # Graficar los resultados

Cell In[410], line 20, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     18 def wrapped_func(*args, **kwargs):
     19     self.reset()
---> 20     return func(*args, **kwargs)

Cell In[408], line 52, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     50 def wrapped_func(*args, **kwargs):
     51     self.reset()
---> 52     return func(*args, **kwargs)

Cell In[406], line 22, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     20 def wrapped_func(*args, **kwargs):
     21     self.reset()
---> 22     return func(*args, **kwargs)

Cell In[410], line 20, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     18 def wrapped_func(*args, **kwargs):
     19     self.reset()
---> 20     return func(*args, **kwargs)
```

```
File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:502, in differential_evolution(func, bounds, args, strategy, maxiter,␣
↪popsize, tol, mutation, recombination, seed, callback, disp, polish, init,␣
↪atol, updating, workers, constraints, x0, integrality, vectorized)
    485 # using a context manager means that any created Pool objects are
    486 # cleared up.
    487 with DifferentialEvolutionSolver(func, bounds, args=args,
    488                                  strategy=strategy,
    489                                  maxiter=maxiter,
   (...)
    500                                  integrality=integrality,
    501                                  vectorized=vectorized) as solver:
--> 502     ret = solver.solve()
    504 return ret

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:1164, in DifferentialEvolutionSolver.solve(self)
   1161 for nit in range(1, self.maxiter + 1):
   1162     # evolve the population by a generation
   1163     try:
-> 1164         next(self)
   1165     except StopIteration:
   1166         warning_flag = True

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:1557, in DifferentialEvolutionSolver.__next__(self)
   1554     raise StopIteration
   1556 # create a trial solution
-> 1557 trial = self._mutate(candidate)
   1559 # ensuring that it's in the range [0, 1)
   1560 self._ensure_constraint(trial)

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:1682, in DifferentialEvolutionSolver._mutate(self, candidate)
   1679     return self._unscale_parameters(trial)
   1681 trial = np.copy(self.population[candidate])
-> 1682 fill_point = rng.choice(self.parameter_count)
   1684 if self.strategy in ['currenttobest1exp', 'currenttobest1bin']:
   1685     bprime = self.mutation_func(candidate,
   1686                                 self._select_samples(candidate, 5))

KeyboardInterrupt:
```

[ ]:
```
V_lig = 0.6
bounds = [(F_0-0.6, F_0) for _ in range(n-2)]


sol_de_vol06 = differential_evolution(
    area_func,
    bounds=bounds,
    constraints=cons_de,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=0.1,
    # disp=True,
    callback=logger
)
Famp_vol06 = np.concatenate(([F0], sol_de_vol06.x, [F1]))
print(vol_func(Famp_vol06))
```

```
Iteración: 1, Valor de la función: 3.428097764338859
Iteración: 2, Valor de la función: 3.947719111159523
Iteración: 3, Valor de la función: 3.8322114514220407
Iteración: 4, Valor de la función: 3.8322114514220407
Iteración: 5, Valor de la función: 2.278244977289836
Iteración: 6, Valor de la función: 2.278244977289836
Iteración: 7, Valor de la función: 1.7881776543695729
Iteración: 8, Valor de la función: 1.7881776543695729
Iteración: 9, Valor de la función: 1.7881776543695729
Iteración: 10, Valor de la función: 1.6578356696493568
Iteración: 11, Valor de la función: 1.4594359273220623
Iteración: 12, Valor de la función: 1.4594359273220623
Iteración: 13, Valor de la función: 1.4594359273220623
Iteración: 14, Valor de la función: 1.4464387732104487
Iteración: 15, Valor de la función: 1.4464387732104487
Iteración: 16, Valor de la función: 1.4464387732104487
Iteración: 17, Valor de la función: 1.4464387732104487
Iteración: 18, Valor de la función: 1.4464387732104487
Iteración: 19, Valor de la función: 1.4464387732104487
Iteración: 20, Valor de la función: 1.4464387732104487
Iteración: 21, Valor de la función: 1.4464387732104487
Iteración: 22, Valor de la función: 1.3254806189593302
Iteración: 23, Valor de la función: 1.3254806189593302
Iteración: 24, Valor de la función: 1.3254806189593302
Iteración: 25, Valor de la función: 1.314951107993129
Iteración: 26, Valor de la función: 1.314951107993129
Iteración: 27, Valor de la función: 1.311554188097202
Iteración: 28, Valor de la función: 1.311554188097202
```

```
Iteración: 29, Valor de la función: 1.311554188097202
Iteración: 30, Valor de la función: 1.311554188097202
Iteración: 31, Valor de la función: 1.311554188097202
Iteración: 32, Valor de la función: 1.3054332797449433
Iteración: 33, Valor de la función: 1.2996506078195038
Iteración: 34, Valor de la función: 1.2996506078195038
Iteración: 35, Valor de la función: 1.2603169585723317
Iteración: 36, Valor de la función: 1.2155143771999681
Iteración: 37, Valor de la función: 1.2155143771999681
Iteración: 38, Valor de la función: 1.2155143771999681
Iteración: 39, Valor de la función: 1.2155143771999681
Iteración: 40, Valor de la función: 1.2155143771999681
Iteración: 41, Valor de la función: 1.2155143771999681
Iteración: 42, Valor de la función: 1.2155143771999681
Iteración: 43, Valor de la función: 1.2155143771999681
Iteración: 44, Valor de la función: 1.2155143771999681
Iteración: 45, Valor de la función: 1.2155143771999681
Iteración: 46, Valor de la función: 1.2155143771999681
Iteración: 47, Valor de la función: 1.2155143771999681
Iteración: 48, Valor de la función: 1.2155143771999681
Iteración: 49, Valor de la función: 1.2155143771999681
Iteración: 50, Valor de la función: 1.2155143771999681
Iteración: 51, Valor de la función: 1.2075382986543426
Iteración: 52, Valor de la función: 1.178001628459669
Iteración: 53, Valor de la función: 1.178001628459669
Iteración: 54, Valor de la función: 1.1650818366365012
Iteración: 55, Valor de la función: 1.1650818366365012
Iteración: 56, Valor de la función: 1.1650818366365012
Iteración: 57, Valor de la función: 1.1650818366365012
Iteración: 58, Valor de la función: 1.1650818366365012
Iteración: 59, Valor de la función: 1.1650818366365012
Iteración: 60, Valor de la función: 1.1650818366365012
Iteración: 61, Valor de la función: 1.1650818366365012
Iteración: 62, Valor de la función: 1.1650818366365012
Iteración: 63, Valor de la función: 1.1430864506712604
Iteración: 64, Valor de la función: 1.1430864506712604
Iteración: 65, Valor de la función: 1.1430864506712604
Iteración: 66, Valor de la función: 1.0754136730738117
Iteración: 67, Valor de la función: 1.0754136730738117
0.047142701108657925
```

[ ]:
```python
V_lig = 1
bounds = [(F_0-0.5, F_0+0.5) for _ in range(n-2)]


sol_de_vol1 = differential_evolution(
    area_func,
```

```python
    bounds=bounds,
    constraints=cons_de,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=0.1,
    # disp=True,
    callback=logger
)
Famp_vol1 = np.concatenate(([F0], sol_de_vol1.x, [F1]))
vol_func(Famp_vol1)
```

```
Iteración: 1, Valor de la función: 3.52713611186477
Iteración: 2, Valor de la función: 3.488479911684285
Iteración: 3, Valor de la función: 3.4075305130398648
Iteración: 4, Valor de la función: 3.3386484622091164
Iteración: 5, Valor de la función: 3.3386484622091164
Iteración: 6, Valor de la función: 3.271146696162755
Iteración: 7, Valor de la función: 3.1530169358437052
Iteración: 8, Valor de la función: 3.1530169358437052
Iteración: 9, Valor de la función: 3.1530169358437052
Iteración: 10, Valor de la función: 3.129813534061818
Iteración: 11, Valor de la función: 2.9696198102232074
Iteración: 12, Valor de la función: 2.6568369000165784
Iteración: 13, Valor de la función: 2.6568369000165784
Iteración: 14, Valor de la función: 2.6568369000165784
Iteración: 15, Valor de la función: 2.2658865240175246
Iteración: 16, Valor de la función: 2.2658865240175246
Iteración: 17, Valor de la función: 2.2658865240175246
Iteración: 18, Valor de la función: 2.2658865240175246
Iteración: 19, Valor de la función: 2.2658865240175246
Iteración: 20, Valor de la función: 2.2658865240175246
Iteración: 21, Valor de la función: 2.2658865240175246
Iteración: 22, Valor de la función: 2.2658865240175246
Iteración: 23, Valor de la función: 2.2658865240175246
Iteración: 24, Valor de la función: 2.2658865240175246
Iteración: 25, Valor de la función: 2.165893387784659
Iteración: 26, Valor de la función: 2.165893387784659
Iteración: 27, Valor de la función: 2.165893387784659
Iteración: 28, Valor de la función: 2.165893387784659
Iteración: 29, Valor de la función: 2.165893387784659
Iteración: 30, Valor de la función: 2.165893387784659
Iteración: 31, Valor de la función: 2.165893387784659
Iteración: 32, Valor de la función: 2.165893387784659
Iteración: 33, Valor de la función: 2.165893387784659
Iteración: 34, Valor de la función: 2.165893387784659
```

```
Iteración: 35, Valor de la función: 2.165893387784659
Iteración: 36, Valor de la función: 2.165893387784659
Iteración: 37, Valor de la función: 2.165893387784659
Iteración: 38, Valor de la función: 2.165893387784659
Iteración: 39, Valor de la función: 2.165893387784659
Iteración: 40, Valor de la función: 2.165893387784659
Iteración: 41, Valor de la función: 2.165893387784659
Iteración: 42, Valor de la función: 2.165893387784659
Iteración: 43, Valor de la función: 2.165893387784659
Iteración: 44, Valor de la función: 2.165893387784659
Iteración: 45, Valor de la función: 2.165893387784659
Iteración: 46, Valor de la función: 2.165893387784659
Iteración: 47, Valor de la función: 2.165893387784659
Iteración: 48, Valor de la función: 2.165893387784659
Iteración: 49, Valor de la función: 2.165893387784659
Iteración: 50, Valor de la función: 2.165893387784659
Iteración: 51, Valor de la función: 2.165893387784659
Iteración: 52, Valor de la función: 2.165893387784659
Iteración: 53, Valor de la función: 2.165893387784659
Iteración: 54, Valor de la función: 1.9713965468470231
Iteración: 55, Valor de la función: 1.9205330808287961
Iteración: 56, Valor de la función: 1.9205330808287961
Iteración: 57, Valor de la función: 1.9205330808287961
Iteración: 58, Valor de la función: 1.8797961837045831
Iteración: 59, Valor de la función: 1.8797961837045831
Iteración: 60, Valor de la función: 1.8797961837045831
Iteración: 61, Valor de la función: 1.8797961837045831
Iteración: 62, Valor de la función: 1.8797961837045831
Iteración: 63, Valor de la función: 1.8797961837045831
Iteración: 64, Valor de la función: 1.7639500041748843
Iteración: 65, Valor de la función: 1.7639500041748843
Iteración: 66, Valor de la función: 1.7639500041748843
Iteración: 67, Valor de la función: 1.7639500041748843
Iteración: 68, Valor de la función: 1.7639500041748843
Iteración: 69, Valor de la función: 1.4694375470419214
Iteración: 70, Valor de la función: 1.4694375470419214
Iteración: 71, Valor de la función: 1.4694375470419214
Iteración: 72, Valor de la función: 1.4694375470419214
Iteración: 73, Valor de la función: 1.4694375470419214
Iteración: 74, Valor de la función: 1.4694375470419214
Iteración: 75, Valor de la función: 1.4694375470419214
Iteración: 76, Valor de la función: 1.4694375470419214
Iteración: 77, Valor de la función: 1.4694375470419214
Iteración: 78, Valor de la función: 1.4694375470419214
Iteración: 79, Valor de la función: 1.4694375470419214
Iteración: 80, Valor de la función: 1.4694375470419214
Iteración: 81, Valor de la función: 1.4694375470419214
Iteración: 82, Valor de la función: 1.4694375470419214
```

```
Iteración: 83, Valor de la función: 1.4694375470419214
Iteración: 84, Valor de la función: 1.398397910885044
Iteración: 85, Valor de la función: 1.398397910885044
Iteración: 86, Valor de la función: 1.398397910885044
Iteración: 87, Valor de la función: 1.398397910885044
Iteración: 88, Valor de la función: 1.398397910885044
Iteración: 89, Valor de la función: 1.398397910885044
Iteración: 90, Valor de la función: 1.398397910885044
Iteración: 91, Valor de la función: 1.398397910885044
Iteración: 92, Valor de la función: 1.398397910885044
Iteración: 93, Valor de la función: 1.398397910885044
Iteración: 94, Valor de la función: 1.398397910885044
Iteración: 95, Valor de la función: 1.398397910885044
Iteración: 96, Valor de la función: 1.398397910885044
Iteración: 97, Valor de la función: 1.398397910885044
Iteración: 98, Valor de la función: 1.398397910885044
Iteración: 99, Valor de la función: 1.398397910885044
Iteración: 100, Valor de la función: 1.398397910885044
Iteración: 101, Valor de la función: 1.398397910885044
Iteración: 102, Valor de la función: 1.398397910885044
Iteración: 103, Valor de la función: 1.398397910885044
Iteración: 104, Valor de la función: 1.398397910885044
Iteración: 105, Valor de la función: 1.398397910885044
Iteración: 106, Valor de la función: 1.2939105545850944
Iteración: 107, Valor de la función: 1.2939105545850944
Iteración: 108, Valor de la función: 1.2939105545850944
Iteración: 109, Valor de la función: 1.2939105545850944
Iteración: 110, Valor de la función: 1.2939105545850944
Iteración: 111, Valor de la función: 1.2939105545850944
Iteración: 112, Valor de la función: 1.2939105545850944
Iteración: 113, Valor de la función: 1.2823829980592338
Iteración: 114, Valor de la función: 1.2046926819332018
Iteración: 115, Valor de la función: 1.2046926819332018
Iteración: 116, Valor de la función: 1.2046926819332018
Iteración: 117, Valor de la función: 1.2046926819332018
Iteración: 118, Valor de la función: 1.2046926819332018
Iteración: 119, Valor de la función: 1.2046926819332018
Iteración: 120, Valor de la función: 1.2046926819332018
Iteración: 121, Valor de la función: 1.2046926819332018
Iteración: 122, Valor de la función: 1.1541592885552328
Iteración: 123, Valor de la función: 1.1541592885552328
Iteración: 124, Valor de la función: 1.1541592885552328
Iteración: 125, Valor de la función: 1.1541592885552328
Iteración: 126, Valor de la función: 1.1221496096402854
Iteración: 127, Valor de la función: 1.1221496096402854
Iteración: 128, Valor de la función: 1.1221496096402854
Iteración: 129, Valor de la función: 1.1221496096402854
Iteración: 130, Valor de la función: 1.1217498128249817
```

```
Iteración: 131, Valor de la función: 1.1004640376212969
Iteración: 132, Valor de la función: 1.1004640376212969
Iteración: 133, Valor de la función: 1.1004640376212969
Iteración: 134, Valor de la función: 1.0971992362664735
Iteración: 135, Valor de la función: 1.0926372821148185
Iteración: 136, Valor de la función: 1.0926372821148185
Iteración: 137, Valor de la función: 1.0926372821148185
Iteración: 138, Valor de la función: 1.0922476696459298
Iteración: 139, Valor de la función: 1.0922476696459298
Iteración: 140, Valor de la función: 1.0922476696459298
Iteración: 141, Valor de la función: 1.0922476696459298
Iteración: 142, Valor de la función: 1.0764583967621715
Iteración: 143, Valor de la función: 1.0764583967621715
Iteración: 144, Valor de la función: 1.0658822548789524
Iteración: 145, Valor de la función: 1.0658822548789524
Iteración: 146, Valor de la función: 1.0598692129686615
Iteración: 147, Valor de la función: 1.0598692129686615
Iteración: 148, Valor de la función: 1.0598573412528687
Iteración: 149, Valor de la función: 1.0598573412528687
Iteración: 150, Valor de la función: 1.0488213559312396
Iteración: 151, Valor de la función: 1.0488213559312396
Iteración: 152, Valor de la función: 1.0488213559312396
Iteración: 153, Valor de la función: 1.0488213559312396
Iteración: 154, Valor de la función: 1.0488213559312396
Iteración: 155, Valor de la función: 1.0315292055173322
Iteración: 156, Valor de la función: 1.0315292055173322
Iteración: 157, Valor de la función: 1.0315292055173322
Iteración: 158, Valor de la función: 1.0315292055173322
Iteración: 159, Valor de la función: 1.0315292055173322
Iteración: 160, Valor de la función: 1.0302273071830688
Iteración: 161, Valor de la función: 1.0302273071830688
Iteración: 162, Valor de la función: 1.02167880570836
Iteración: 163, Valor de la función: 1.0162537509285638
Iteración: 164, Valor de la función: 1.0162537509285638
Iteración: 165, Valor de la función: 1.0152675129147903
Iteración: 166, Valor de la función: 1.0152675129147903
Iteración: 167, Valor de la función: 1.0152675129147903
Iteración: 168, Valor de la función: 1.012606971645115
Iteración: 169, Valor de la función: 1.012606971645115
Iteración: 170, Valor de la función: 1.0046312515533915
Iteración: 171, Valor de la función: 1.0046312515533915
Iteración: 172, Valor de la función: 1.0046312515533915
Iteración: 173, Valor de la función: 1.0036578609912716
Iteración: 174, Valor de la función: 1.0011561810083562
```

[ ]: -0.009047599453626542

[ ]:

```python
V_lig = 1.5
bounds = [(F_0, F_0)+2 for _ in range(n-2)]


sol_de_vol15 = differential_evolution(
    area_func,
    bounds=bounds,
    constraints=cons_de,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=0.1,
    # disp=True,
    callback=logger
)
Famp_vol15 = np.concatenate(([F0], sol_de_vol15.x, [F1]))
vol_func(Famp_vol15)
```

```
Iteración: 1, Valor de la función: 5.894985601885647
Iteración: 2, Valor de la función: 4.80182176885743
Iteración: 3, Valor de la función: 4.1091486366868155
Iteración: 4, Valor de la función: 4.1091486366868155
Iteración: 5, Valor de la función: 3.820321561869781
Iteración: 6, Valor de la función: 3.820321561869781
Iteración: 7, Valor de la función: 3.820321561869781
Iteración: 8, Valor de la función: 3.4427689618069834
Iteración: 9, Valor de la función: 3.4427689618069834
Iteración: 10, Valor de la función: 3.4427689618069834
Iteración: 11, Valor de la función: 3.4427689618069834
Iteración: 12, Valor de la función: 3.3633665937011448
Iteración: 13, Valor de la función: 3.002418177136725
Iteración: 14, Valor de la función: 2.9590036908875024
Iteración: 15, Valor de la función: 2.9590036908875024
Iteración: 16, Valor de la función: 2.9590036908875024
Iteración: 17, Valor de la función: 2.9590036908875024
Iteración: 18, Valor de la función: 2.9590036908875024
Iteración: 19, Valor de la función: 2.8423245102648727
Iteración: 20, Valor de la función: 2.8423245102648727
Iteración: 21, Valor de la función: 2.8423245102648727
Iteración: 22, Valor de la función: 2.8423245102648727
Iteración: 23, Valor de la función: 2.61180674372982
Iteración: 24, Valor de la función: 2.61180674372982
Iteración: 25, Valor de la función: 2.61180674372982
Iteración: 26, Valor de la función: 2.61180674372982
Iteración: 27, Valor de la función: 2.61180674372982
Iteración: 28, Valor de la función: 2.503233986055357
```

```
Iteración: 29, Valor de la función: 2.492332672678703
Iteración: 30, Valor de la función: 2.492332672678703
Iteración: 31, Valor de la función: 2.492332672678703
Iteración: 32, Valor de la función: 2.492332672678703
Iteración: 33, Valor de la función: 2.492332672678703
Iteración: 34, Valor de la función: 2.492332672678703
Iteración: 35, Valor de la función: 2.492332672678703
Iteración: 36, Valor de la función: 2.492332672678703
Iteración: 37, Valor de la función: 2.492332672678703
Iteración: 38, Valor de la función: 2.492332672678703
Iteración: 39, Valor de la función: 2.492332672678703
Iteración: 40, Valor de la función: 2.483746138323026
Iteración: 41, Valor de la función: 2.483746138323026
Iteración: 42, Valor de la función: 2.474378252772908
Iteración: 43, Valor de la función: 2.474378252772908
Iteración: 44, Valor de la función: 2.474378252772908
Iteración: 45, Valor de la función: 2.474378252772908
Iteración: 46, Valor de la función: 2.474378252772908
Iteración: 47, Valor de la función: 2.1899701342219333
Iteración: 48, Valor de la función: 2.1899701342219333
Iteración: 49, Valor de la función: 2.1899701342219333
Iteración: 50, Valor de la función: 2.1899701342219333
Iteración: 51, Valor de la función: 2.1899701342219333
Iteración: 52, Valor de la función: 2.1899701342219333
Iteración: 53, Valor de la función: 2.1899701342219333
Iteración: 54, Valor de la función: 2.1899701342219333
Iteración: 55, Valor de la función: 2.1664353530613565
Iteración: 56, Valor de la función: 2.1664353530613565
Iteración: 57, Valor de la función: 2.1664353530613565
Iteración: 58, Valor de la función: 2.1664353530613565
Iteración: 59, Valor de la función: 2.062212554128064
Iteración: 60, Valor de la función: 2.062212554128064
Iteración: 61, Valor de la función: 2.062212554128064
Iteración: 62, Valor de la función: 2.062212554128064
Iteración: 63, Valor de la función: 2.062212554128064
Iteración: 64, Valor de la función: 2.062212554128064
Iteración: 65, Valor de la función: 2.062212554128064
Iteración: 66, Valor de la función: 2.062212554128064
Iteración: 67, Valor de la función: 2.062212554128064
Iteración: 68, Valor de la función: 2.062212554128064
Iteración: 69, Valor de la función: 2.062212554128064
Iteración: 70, Valor de la función: 2.062212554128064
Iteración: 71, Valor de la función: 2.062212554128064
Iteración: 72, Valor de la función: 2.062212554128064
Iteración: 73, Valor de la función: 2.062212554128064
Iteración: 74, Valor de la función: 2.062212554128064
Iteración: 75, Valor de la función: 2.062212554128064
Iteración: 76, Valor de la función: 1.9615040966737705
```

```
Iteración: 77, Valor de la función: 1.9615040966737705
Iteración: 78, Valor de la función: 1.9615040966737705
Iteración: 79, Valor de la función: 1.8786909217105423
Iteración: 80, Valor de la función: 1.8786909217105423
Iteración: 81, Valor de la función: 1.8786909217105423
Iteración: 82, Valor de la función: 1.8786909217105423
Iteración: 83, Valor de la función: 1.8786909217105423
Iteración: 84, Valor de la función: 1.8786909217105423
Iteración: 85, Valor de la función: 1.8786909217105423
Iteración: 86, Valor de la función: 1.8786909217105423
Iteración: 87, Valor de la función: 1.8786909217105423
Iteración: 88, Valor de la función: 1.8392211758137647
Iteración: 89, Valor de la función: 1.8392211758137647
Iteración: 90, Valor de la función: 1.8392211758137647
Iteración: 91, Valor de la función: 1.8392211758137647
Iteración: 92, Valor de la función: 1.8392211758137647
Iteración: 93, Valor de la función: 1.8392211758137647
Iteración: 94, Valor de la función: 1.8392211758137647
Iteración: 95, Valor de la función: 1.804807891584016
Iteración: 96, Valor de la función: 1.804807891584016
Iteración: 97, Valor de la función: 1.804807891584016
Iteración: 98, Valor de la función: 1.804807891584016
Iteración: 99, Valor de la función: 1.804807891584016
Iteración: 100, Valor de la función: 1.804807891584016
Iteración: 101, Valor de la función: 1.7052196065377332
Iteración: 102, Valor de la función: 1.7052196065377332
Iteración: 103, Valor de la función: 1.7052196065377332
Iteración: 104, Valor de la función: 1.6285695308203134
Iteración: 105, Valor de la función: 1.6285695308203134
Iteración: 106, Valor de la función: 1.6285695308203134
Iteración: 107, Valor de la función: 1.6285695308203134
Iteración: 108, Valor de la función: 1.6285695308203134
Iteración: 109, Valor de la función: 1.6285695308203134
Iteración: 110, Valor de la función: 1.6285695308203134
Iteración: 111, Valor de la función: 1.6285695308203134
Iteración: 112, Valor de la función: 1.6285695308203134
Iteración: 113, Valor de la función: 1.6285695308203134
Iteración: 114, Valor de la función: 1.6285695308203134
Iteración: 115, Valor de la función: 1.6285695308203134
Iteración: 116, Valor de la función: 1.6285695308203134
Iteración: 117, Valor de la función: 1.6086762052761052
Iteración: 118, Valor de la función: 1.6086762052761052
Iteración: 119, Valor de la función: 1.6086762052761052
Iteración: 120, Valor de la función: 1.6086762052761052
Iteración: 121, Valor de la función: 1.6086762052761052
Iteración: 122, Valor de la función: 1.6086762052761052
Iteración: 123, Valor de la función: 1.6086762052761052
Iteración: 124, Valor de la función: 1.6086762052761052
```

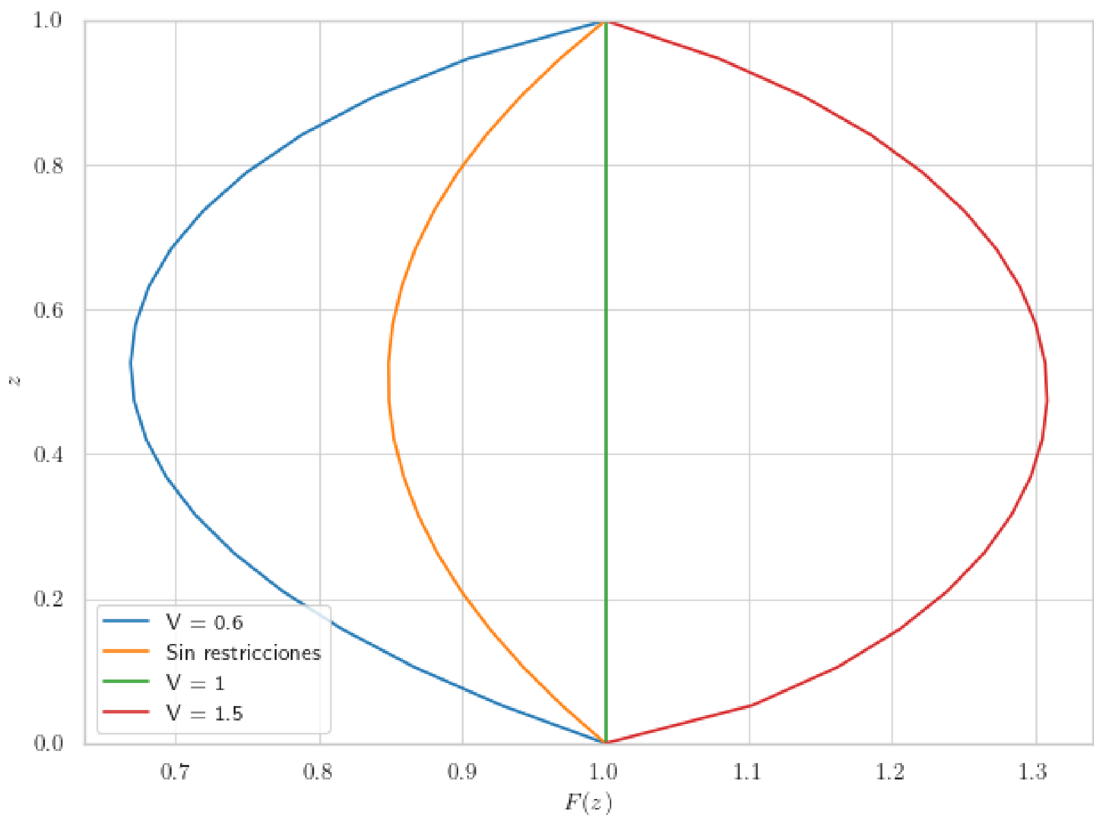
```
Iteración: 125, Valor de la función: 1.6086762052761052
Iteración: 126, Valor de la función: 1.6086762052761052
Iteración: 127, Valor de la función: 1.6086762052761052
Iteración: 128, Valor de la función: 1.6086762052761052
Iteración: 129, Valor de la función: 1.6086762052761052
Iteración: 130, Valor de la función: 1.6086762052761052
Iteración: 131, Valor de la función: 1.6086762052761052
Iteración: 132, Valor de la función: 1.6086762052761052
Iteración: 133, Valor de la función: 1.6086762052761052
Iteración: 134, Valor de la función: 1.6086762052761052
Iteración: 135, Valor de la función: 1.6086762052761052
Iteración: 136, Valor de la función: 1.6086762052761052
Iteración: 137, Valor de la función: 1.6086762052761052
Iteración: 138, Valor de la función: 1.6086762052761052
Iteración: 139, Valor de la función: 1.579618463553598
Iteración: 140, Valor de la función: 1.5705322885294628
Iteración: 141, Valor de la función: 1.5431166512199093
Iteración: 142, Valor de la función: 1.5431166512199093
Iteración: 143, Valor de la función: 1.5431166512199093
Iteración: 144, Valor de la función: 1.5431166512199093
Iteración: 145, Valor de la función: 1.5431166512199093
Iteración: 146, Valor de la función: 1.5431166512199093
Iteración: 147, Valor de la función: 1.5431166512199093
Iteración: 148, Valor de la función: 1.5428161240850704
Iteración: 149, Valor de la función: 1.5428161240850704
Iteración: 150, Valor de la función: 1.536359091468596
Iteración: 151, Valor de la función: 1.536359091468596
Iteración: 152, Valor de la función: 1.536359091468596
Iteración: 153, Valor de la función: 1.536359091468596
Iteración: 154, Valor de la función: 1.536359091468596
Iteración: 155, Valor de la función: 1.536359091468596
Iteración: 156, Valor de la función: 1.5288138183342719
Iteración: 157, Valor de la función: 1.5288138183342719
Iteración: 158, Valor de la función: 1.5267075123427813
Iteración: 159, Valor de la función: 1.5168919304790045
Iteración: 160, Valor de la función: 1.5168919304790045
Iteración: 161, Valor de la función: 1.5168919304790045
Iteración: 162, Valor de la función: 1.5168919304790045
Iteración: 163, Valor de la función: 1.5168919304790045
Iteración: 164, Valor de la función: 1.5168919304790045
Iteración: 165, Valor de la función: 1.5168919304790045
Iteración: 166, Valor de la función: 1.5168919304790045
Iteración: 167, Valor de la función: 1.5168919304790045
Iteración: 168, Valor de la función: 1.5168919304790045
Iteración: 169, Valor de la función: 1.5168919304790045
Iteración: 170, Valor de la función: 1.5168919304790045
Iteración: 171, Valor de la función: 1.5168919304790045
Iteración: 172, Valor de la función: 1.5168919304790045
```

```
Iteración: 173, Valor de la función: 1.5168919304790045
Iteración: 174, Valor de la función: 1.5168919304790045
Iteración: 175, Valor de la función: 1.5168919304790045
Iteración: 176, Valor de la función: 1.5168919304790045
Iteración: 177, Valor de la función: 1.5168919304790045
Iteración: 178, Valor de la función: 1.5168919304790045
Iteración: 179, Valor de la función: 1.5168919304790045
Iteración: 180, Valor de la función: 1.5168919304790045
Iteración: 181, Valor de la función: 1.5168919304790045
Iteración: 182, Valor de la función: 1.515104964002688
Iteración: 183, Valor de la función: 1.515104964002688
Iteración: 184, Valor de la función: 1.5006895881905407
Iteración: 185, Valor de la función: 1.5006895881905407
Iteración: 186, Valor de la función: 1.5006895881905407
Iteración: 187, Valor de la función: 1.5006895881905407
Iteración: 188, Valor de la función: 1.5006895881905407
Iteración: 189, Valor de la función: 1.5006895881905407
Iteración: 190, Valor de la función: 1.5006895881905407
Iteración: 191, Valor de la función: 1.5006895881905407
Iteración: 192, Valor de la función: 1.5006895881905407
Iteración: 193, Valor de la función: 1.5006895881905407
Iteración: 194, Valor de la función: 1.5006895881905407
Iteración: 195, Valor de la función: 1.5006895881905407
Iteración: 196, Valor de la función: 1.5006895881905407
Iteración: 197, Valor de la función: 1.5006895881905407
Iteración: 198, Valor de la función: 1.5006895881905407
Iteración: 199, Valor de la función: 1.5006895881905407
Iteración: 200, Valor de la función: 1.4977525471241813
Iteración: 201, Valor de la función: 1.4975515566614166
Iteración: 202, Valor de la función: 1.4975515566614166
Iteración: 203, Valor de la función: 1.4975515566614166
Iteración: 204, Valor de la función: 1.4975515566614166
Iteración: 205, Valor de la función: 1.4943090600973594
Iteración: 206, Valor de la función: 1.4943090600973594
Iteración: 207, Valor de la función: 1.4943090600973594
Iteración: 208, Valor de la función: 1.4943090600973594
Iteración: 209, Valor de la función: 1.4943090600973594
Iteración: 210, Valor de la función: 1.4943090600973594
Iteración: 211, Valor de la función: 1.4896732223353053
Iteración: 212, Valor de la función: 1.4896732223353053
Iteración: 213, Valor de la función: 1.4896732223353053
Iteración: 214, Valor de la función: 1.48805296747537
```

[ ]: -0.0566666293907605

[ ]:
```
plt.plot(Famp_vol06, z, label='V = 0.6')
plt.plot(Famp_vol_de, z, label='Sin restricciones')
```

```
plt.plot(Famp_vol1, z, label='V = 1')
plt.plot(Famp_vol15, z, label='V = 1.5')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.legend()
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.title('Curva optimizada con restricción de volumen')
plt.savefig('Figuras/vol_inic_de.pdf', format='pdf')
plt.show()
```



[ ]:
```
V_lig = 1.6
bounds = [(F_0, F_0+1) for _ in range(n-2)]


sol_grad_vol16 = differential_evolution(
    area_func,
    bounds=bounds,
    constraints=cons_de,
```

```python
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=0.01,
    # disp=True,
    callback=logger
)
Famp_vol16 = np.concatenate(([F0], sol_grad_vol16.x, [F1]))
```

```
Iteración: 1, Valor de la función: 5.9167027620975885
Iteración: 2, Valor de la función: 5.010386835050035
Iteración: 3, Valor de la función: 5.010386835050035
Iteración: 4, Valor de la función: 3.919130559901436
Iteración: 5, Valor de la función: 3.919130559901436
Iteración: 6, Valor de la función: 3.919130559901436
Iteración: 7, Valor de la función: 3.919130559901436
Iteración: 8, Valor de la función: 3.919130559901436
Iteración: 9, Valor de la función: 3.919130559901436
Iteración: 10, Valor de la función: 3.919130559901436
Iteración: 11, Valor de la función: 3.919130559901436
Iteración: 12, Valor de la función: 3.7801014228735728
Iteración: 13, Valor de la función: 3.1545737907991063
Iteración: 14, Valor de la función: 2.97364355439043
Iteración: 15, Valor de la función: 2.97364355439043
Iteración: 16, Valor de la función: 2.97364355439043
Iteración: 17, Valor de la función: 2.97364355439043
Iteración: 18, Valor de la función: 2.97364355439043
Iteración: 19, Valor de la función: 2.97364355439043
Iteración: 20, Valor de la función: 2.97364355439043
Iteración: 21, Valor de la función: 2.97364355439043
Iteración: 22, Valor de la función: 2.5459631380659666
Iteración: 23, Valor de la función: 2.5459631380659666
Iteración: 24, Valor de la función: 2.5459631380659666
Iteración: 25, Valor de la función: 2.5459631380659666
Iteración: 26, Valor de la función: 2.5459631380659666
Iteración: 27, Valor de la función: 2.5459631380659666
Iteración: 28, Valor de la función: 2.5459631380659666
Iteración: 29, Valor de la función: 2.500520268324926
Iteración: 30, Valor de la función: 2.500520268324926
Iteración: 31, Valor de la función: 2.500520268324926
Iteración: 32, Valor de la función: 2.500520268324926
Iteración: 33, Valor de la función: 2.2020948361624764
Iteración: 34, Valor de la función: 2.2020948361624764
Iteración: 35, Valor de la función: 2.2020948361624764
Iteración: 36, Valor de la función: 2.2020948361624764
Iteración: 37, Valor de la función: 2.2020948361624764
```

```
Iteración: 38, Valor de la función: 2.2020948361624764
Iteración: 39, Valor de la función: 2.2020948361624764
Iteración: 40, Valor de la función: 2.2020948361624764
Iteración: 41, Valor de la función: 2.2020948361624764
Iteración: 42, Valor de la función: 2.2020948361624764
Iteración: 43, Valor de la función: 2.2020948361624764
Iteración: 44, Valor de la función: 2.2020948361624764
Iteración: 45, Valor de la función: 2.2020948361624764
Iteración: 46, Valor de la función: 2.2020948361624764
Iteración: 47, Valor de la función: 2.2020948361624764
Iteración: 48, Valor de la función: 2.2020948361624764
Iteración: 49, Valor de la función: 2.2020948361624764
Iteración: 50, Valor de la función: 2.2020948361624764
Iteración: 51, Valor de la función: 2.2020948361624764
Iteración: 52, Valor de la función: 2.2020948361624764
Iteración: 53, Valor de la función: 2.2020948361624764
Iteración: 54, Valor de la función: 2.2020948361624764
Iteración: 55, Valor de la función: 2.2020948361624764
Iteración: 56, Valor de la función: 2.2020948361624764
Iteración: 57, Valor de la función: 2.2020948361624764
Iteración: 58, Valor de la función: 2.2020948361624764
Iteración: 59, Valor de la función: 2.2020948361624764
Iteración: 60, Valor de la función: 2.2020948361624764
Iteración: 61, Valor de la función: 2.2020948361624764
Iteración: 62, Valor de la función: 2.2020948361624764
Iteración: 63, Valor de la función: 2.2020948361624764
Iteración: 64, Valor de la función: 2.2020948361624764
Iteración: 65, Valor de la función: 2.2020948361624764
Iteración: 66, Valor de la función: 2.1186498347151623
Iteración: 67, Valor de la función: 2.1186498347151623
Iteración: 68, Valor de la función: 2.0995221302177596
Iteración: 69, Valor de la función: 2.0995221302177596
Iteración: 70, Valor de la función: 2.0995221302177596
Iteración: 71, Valor de la función: 2.0995221302177596
Iteración: 72, Valor de la función: 2.0995221302177596
Iteración: 73, Valor de la función: 2.0995221302177596
Iteración: 74, Valor de la función: 2.0995221302177596
Iteración: 75, Valor de la función: 2.0995221302177596
Iteración: 76, Valor de la función: 2.0995221302177596
Iteración: 77, Valor de la función: 2.0995221302177596
Iteración: 78, Valor de la función: 2.0995221302177596
Iteración: 79, Valor de la función: 1.878666943121697
Iteración: 80, Valor de la función: 1.878666943121697
Iteración: 81, Valor de la función: 1.878666943121697
Iteración: 82, Valor de la función: 1.878666943121697
Iteración: 83, Valor de la función: 1.878666943121697
Iteración: 84, Valor de la función: 1.878666943121697
Iteración: 85, Valor de la función: 1.878666943121697
```

```
Iteración: 86, Valor de la función: 1.878666943121697
Iteración: 87, Valor de la función: 1.878666943121697
Iteración: 88, Valor de la función: 1.878666943121697
Iteración: 89, Valor de la función: 1.878666943121697
Iteración: 90, Valor de la función: 1.878666943121697
Iteración: 91, Valor de la función: 1.878666943121697
Iteración: 92, Valor de la función: 1.878666943121697
Iteración: 93, Valor de la función: 1.878666943121697
Iteración: 94, Valor de la función: 1.878666943121697
Iteración: 95, Valor de la función: 1.878666943121697
Iteración: 96, Valor de la función: 1.878666943121697
Iteración: 97, Valor de la función: 1.878666943121697
Iteración: 98, Valor de la función: 1.878666943121697
Iteración: 99, Valor de la función: 1.878666943121697
Iteración: 100, Valor de la función: 1.878666943121697
Iteración: 101, Valor de la función: 1.878666943121697
Iteración: 102, Valor de la función: 1.878666943121697
Iteración: 103, Valor de la función: 1.878666943121697
Iteración: 104, Valor de la función: 1.878666943121697
Iteración: 105, Valor de la función: 1.878666943121697
Iteración: 106, Valor de la función: 1.7749657552732407
Iteración: 107, Valor de la función: 1.7749657552732407
Iteración: 108, Valor de la función: 1.7749657552732407
Iteración: 109, Valor de la función: 1.7587211534096014
Iteración: 110, Valor de la función: 1.7587211534096014
Iteración: 111, Valor de la función: 1.7587211534096014
Iteración: 112, Valor de la función: 1.7587211534096014
Iteración: 113, Valor de la función: 1.7587211534096014
Iteración: 114, Valor de la función: 1.6666447414019112
Iteración: 115, Valor de la función: 1.6666447414019112
Iteración: 116, Valor de la función: 1.6666447414019112
Iteración: 117, Valor de la función: 1.6666447414019112
Iteración: 118, Valor de la función: 1.6666447414019112
Iteración: 119, Valor de la función: 1.6666447414019112
Iteración: 120, Valor de la función: 1.6666447414019112
Iteración: 121, Valor de la función: 1.6666447414019112
Iteración: 122, Valor de la función: 1.6666447414019112
Iteración: 123, Valor de la función: 1.6666447414019112
Iteración: 124, Valor de la función: 1.6666447414019112
Iteración: 125, Valor de la función: 1.6666447414019112
Iteración: 126, Valor de la función: 1.6666447414019112
Iteración: 127, Valor de la función: 1.6519562034229094
Iteración: 128, Valor de la función: 1.6519562034229094
Iteración: 129, Valor de la función: 1.6519562034229094
Iteración: 130, Valor de la función: 1.6519562034229094
Iteración: 131, Valor de la función: 1.6519562034229094
Iteración: 132, Valor de la función: 1.6519562034229094
Iteración: 133, Valor de la función: 1.6519562034229094
```

```
Iteración: 134, Valor de la función: 1.6519562034229094
Iteración: 135, Valor de la función: 1.6519562034229094
Iteración: 136, Valor de la función: 1.6519562034229094
Iteración: 137, Valor de la función: 1.6519562034229094
Iteración: 138, Valor de la función: 1.6459679576823731
Iteración: 139, Valor de la función: 1.6459679576823731
Iteración: 140, Valor de la función: 1.6459679576823731
Iteración: 141, Valor de la función: 1.6459679576823731
Iteración: 142, Valor de la función: 1.6459679576823731
Iteración: 143, Valor de la función: 1.6270378603662932
Iteración: 144, Valor de la función: 1.6270378603662932
Iteración: 145, Valor de la función: 1.6270378603662932
Iteración: 146, Valor de la función: 1.6270378603662932
Iteración: 147, Valor de la función: 1.6270378603662932
Iteración: 148, Valor de la función: 1.6270378603662932
Iteración: 149, Valor de la función: 1.6270378603662932
Iteración: 150, Valor de la función: 1.6270378603662932
Iteración: 151, Valor de la función: 1.6270378603662932
Iteración: 152, Valor de la función: 1.6269701476368306
Iteración: 153, Valor de la función: 1.6269701476368306
Iteración: 154, Valor de la función: 1.6209743666711864
Iteración: 155, Valor de la función: 1.6209743666711864
Iteración: 156, Valor de la función: 1.6209743666711864
Iteración: 157, Valor de la función: 1.6192459988546049
Iteración: 158, Valor de la función: 1.6192459988546049
Iteración: 159, Valor de la función: 1.6162654266251455
Iteración: 160, Valor de la función: 1.6162654266251455
Iteración: 161, Valor de la función: 1.6162654266251455
Iteración: 162, Valor de la función: 1.6162654266251455
Iteración: 163, Valor de la función: 1.6141241263310044
Iteración: 164, Valor de la función: 1.6141241263310044
Iteración: 165, Valor de la función: 1.6141241263310044
Iteración: 166, Valor de la función: 1.6141241263310044
Iteración: 167, Valor de la función: 1.6120977818217381
```

[ ]:
```
V_lig = 0.5
bounds = [(F_0-0.7, F_0) for _ in range(n-2)]


sol_grad_vol05 = differential_evolution(
    area_func,
    bounds=bounds,
    constraints=cons_de,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
```

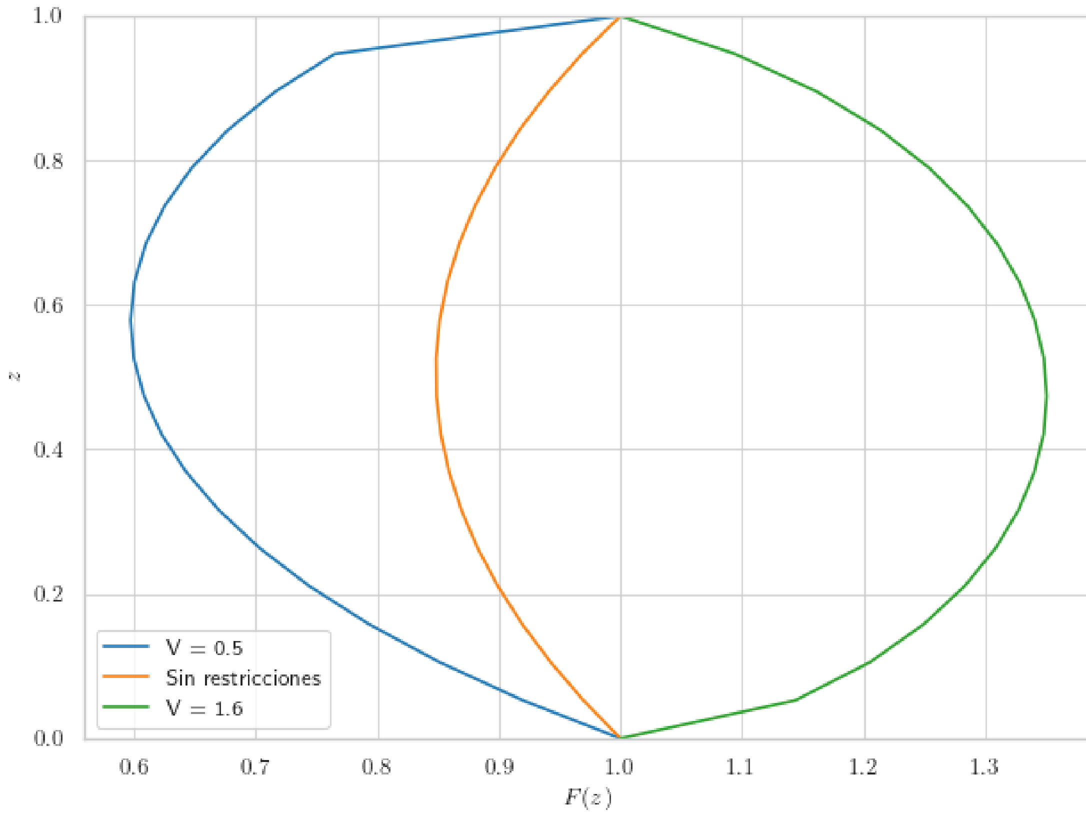
```
    strategy='best1bin',
    tol=0.01,
    # disp=True,
    callback=logger
)
Famp_vol05 = np.concatenate(([F0], sol_grad_vol05.x, [F1]))
```

```
Iteración: 1, Valor de la función: 1.8233323283603604
Iteración: 2, Valor de la función: 1.7839585127271254
Iteración: 3, Valor de la función: 1.7114296950098888
Iteración: 4, Valor de la función: 1.4957197690550772
Iteración: 5, Valor de la función: 1.4957197690550772
Iteración: 6, Valor de la función: 1.4957197690550772
Iteración: 7, Valor de la función: 1.4954540901333555
Iteración: 8, Valor de la función: 1.4954540901333555
Iteración: 9, Valor de la función: 1.4954540901333555
Iteración: 10, Valor de la función: 1.4954540901333555
Iteración: 11, Valor de la función: 1.4954540901333555
Iteración: 12, Valor de la función: 1.4954540901333555
Iteración: 13, Valor de la función: 1.4954540901333555
Iteración: 14, Valor de la función: 1.4954540901333555
Iteración: 15, Valor de la función: 1.4954540901333555
Iteración: 16, Valor de la función: 1.4954540901333555
Iteración: 17, Valor de la función: 1.4954540901333555
Iteración: 18, Valor de la función: 1.4609695638780242
Iteración: 19, Valor de la función: 1.3783683583277275
Iteración: 20, Valor de la función: 1.3783683583277275
Iteración: 21, Valor de la función: 1.3783683583277275
Iteración: 22, Valor de la función: 1.3783683583277275
Iteración: 23, Valor de la función: 1.3265079690758748
Iteración: 24, Valor de la función: 1.3265079690758748
Iteración: 25, Valor de la función: 1.3237142326983142
Iteración: 26, Valor de la función: 1.3237142326983142
Iteración: 27, Valor de la función: 1.3237142326983142
Iteración: 28, Valor de la función: 1.3237142326983142
Iteración: 29, Valor de la función: 1.3237142326983142
Iteración: 30, Valor de la función: 1.3237142326983142
Iteración: 31, Valor de la función: 1.3237142326983142
Iteración: 32, Valor de la función: 1.3237142326983142
Iteración: 33, Valor de la función: 1.3089678391660469
Iteración: 34, Valor de la función: 1.3089678391660469
Iteración: 35, Valor de la función: 1.3089678391660469
Iteración: 36, Valor de la función: 1.3089678391660469
Iteración: 37, Valor de la función: 1.3089678391660469
Iteración: 38, Valor de la función: 1.3089678391660469
Iteración: 39, Valor de la función: 1.3089678391660469
Iteración: 40, Valor de la función: 1.3089678391660469
Iteración: 41, Valor de la función: 1.293774705996881
```

```
Iteración: 42, Valor de la función: 1.293774705996881
Iteración: 43, Valor de la función: 1.293774705996881
Iteración: 44, Valor de la función: 1.293774705996881
Iteración: 45, Valor de la función: 1.2259177464101727
Iteración: 46, Valor de la función: 1.2259177464101727
Iteración: 47, Valor de la función: 1.2259177464101727
Iteración: 48, Valor de la función: 1.2259177464101727
Iteración: 49, Valor de la función: 1.2259177464101727
Iteración: 50, Valor de la función: 1.2259177464101727
Iteración: 51, Valor de la función: 1.2259177464101727
Iteración: 52, Valor de la función: 1.205858631651523
Iteración: 53, Valor de la función: 1.205858631651523
Iteración: 54, Valor de la función: 1.1611510090951689
Iteración: 55, Valor de la función: 1.1611510090951689
Iteración: 56, Valor de la función: 1.1611510090951689
Iteración: 57, Valor de la función: 1.1611510090951689
Iteración: 58, Valor de la función: 1.1611510090951689
Iteración: 59, Valor de la función: 1.1611510090951689
Iteración: 60, Valor de la función: 1.140749885222153
Iteración: 61, Valor de la función: 1.140749885222153
Iteración: 62, Valor de la función: 1.140749885222153
Iteración: 63, Valor de la función: 1.140749885222153
Iteración: 64, Valor de la función: 1.140749885222153
Iteración: 65, Valor de la función: 1.140749885222153
Iteración: 66, Valor de la función: 1.140749885222153
Iteración: 67, Valor de la función: 1.140749885222153
Iteración: 68, Valor de la función: 1.140749885222153
Iteración: 69, Valor de la función: 1.140749885222153
Iteración: 70, Valor de la función: 1.140749885222153
Iteración: 71, Valor de la función: 1.1128010295906714
Iteración: 72, Valor de la función: 1.1128010295906714
Iteración: 73, Valor de la función: 1.1128010295906714
Iteración: 74, Valor de la función: 1.1128010295906714
Iteración: 75, Valor de la función: 1.1128010295906714
Iteración: 76, Valor de la función: 1.1128010295906714
Iteración: 77, Valor de la función: 1.1128010295906714
Iteración: 78, Valor de la función: 1.1128010295906714
Iteración: 79, Valor de la función: 1.1128010295906714
Iteración: 80, Valor de la función: 1.0971940468017083
Iteración: 81, Valor de la función: 1.0966096413967998
Iteración: 82, Valor de la función: 1.0966096413967998
Iteración: 83, Valor de la función: 1.0966096413967998
Iteración: 84, Valor de la función: 1.0900798983172526
Iteración: 85, Valor de la función: 1.0582510359163844
Iteración: 86, Valor de la función: 1.0582510359163844
Iteración: 87, Valor de la función: 1.0582510359163844
Iteración: 88, Valor de la función: 1.0582510359163844
Iteración: 89, Valor de la función: 1.0582510359163844
```

```
Iteración: 90, Valor de la función: 1.0582510359163844
Iteración: 91, Valor de la función: 1.0582510359163844
Iteración: 92, Valor de la función: 1.0582510359163844
Iteración: 93, Valor de la función: 1.0484553586402647
Iteración: 94, Valor de la función: 1.0484553586402647
Iteración: 95, Valor de la función: 1.0484553586402647
Iteración: 96, Valor de la función: 1.0484553586402647
Iteración: 97, Valor de la función: 1.0484553586402647
Iteración: 98, Valor de la función: 1.0484553586402647
Iteración: 99, Valor de la función: 1.0484553586402647
Iteración: 100, Valor de la función: 1.0484553586402647
Iteración: 101, Valor de la función: 1.0484553586402647
Iteración: 102, Valor de la función: 1.0484553586402647
Iteración: 103, Valor de la función: 1.0484553586402647
Iteración: 104, Valor de la función: 1.0484553586402647
Iteración: 105, Valor de la función: 1.0455048934597664
Iteración: 106, Valor de la función: 1.0383693878827673
Iteración: 107, Valor de la función: 1.0383693878827673
Iteración: 108, Valor de la función: 1.0383693878827673
Iteración: 109, Valor de la función: 1.0383693878827673
Iteración: 110, Valor de la función: 1.0383693878827673
Iteración: 111, Valor de la función: 1.0369063516159405
Iteración: 112, Valor de la función: 1.0369063516159405
Iteración: 113, Valor de la función: 1.0369063516159405
Iteración: 114, Valor de la función: 1.0369063516159405
Iteración: 115, Valor de la función: 1.0369063516159405
Iteración: 116, Valor de la función: 1.03328360365418
Iteración: 117, Valor de la función: 1.03328360365418
Iteración: 118, Valor de la función: 1.03328360365418
Iteración: 119, Valor de la función: 1.03328360365418
Iteración: 120, Valor de la función: 1.03328360365418
Iteración: 121, Valor de la función: 1.03328360365418
```

[ ]:
```
plt.plot(Famp_vol05, z, label='V = 0.5')
plt.plot(Famp_vol_de, z, label='Sin restricciones')
plt.plot(Famp_vol16, z, label='V = 1.6')
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.legend()
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.title('Valores inexistentes o inestables de la solución, para $n = {}$'.
↪format(n))
plt.savefig('Figuras/vol_no_sol_de.pdf', format='pdf')
plt.show()
```



1.4.3 Pruebas de casos peculiares

Restricción de la solución igual al volumen original

[250]:
```
F_0 = 1
F0 = F_0
F1 = F_0
n = 20
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

z = np.linspace(0, 1, n)
```

[251]:
```
sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_grad = np.concatenate(([F0], sol_grad.x, [F1]))
sol_grad
```

[251]:
```
 message: Optimization terminated successfully
 success: True
```

```
  status: 0
     fun: 0.9537487850868308
       x: [ 9.697e-01  9.429e-01 ...  9.408e-01  9.684e-01]
     nit: 21
     jac: [-3.795e-04  4.508e-04 ... -3.908e-04  1.709e-04]
    nfev: 429
    njev: 21
```

[252]:
```python
V_lig = 0.0 # la ligadura de volumen

vol_no_cons = vol_func(sol_grad.x)
print('Volumen sin restricciones =', vol_no_cons)
V_lig = vol_no_cons
```

```
Volumen sin restricciones = 0.8095367935945682
```

[253]:
```python
cons = ({'type': 'eq', 'fun': vol_func})
```

Método gradiente

[254]:
```python
sol_grad_cons = minimize(area_func, F_init, constraints=cons, bounds=bounds,␣
 ↪method='SLSQP')
Famp_grad_cons = np.concatenate(([F0], sol_grad_cons.x, [F1]))
sol_grad_cons
```

[254]:
```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 0.9537488908064367
       x: [ 9.696e-01  9.427e-01 ...  9.408e-01  9.685e-01]
     nit: 21
     jac: [-1.110e-04  8.175e-04 ...  2.985e-04 -1.336e-03]
    nfev: 428
    njev: 21
```

[255]:
```python
np.linalg.norm(Famp_grad_cons - Famp_grad)
```

[255]:
```
0.0007030298908543652
```

Algoritmo heurístico

[256]:
```python
bounds = [(F_0-0.5, F_0) for _ in range(n-2)]

sol_de = differential_evolution(
    area_func,
    bounds=bounds,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
```

```python
    recombination=0.7,
    strategy='best1bin',
    tol=0.01,
    # disp=True,
    callback=logger
)
Famp_de = np.concatenate(([F0], sol_de.x, [F1]))
sol_de
```

```
Iteración: 1, Valor de la función: 1.3608089223907232
Iteración: 2, Valor de la función: 1.3608089223907232
Iteración: 3, Valor de la función: 1.3452297724865552
Iteración: 4, Valor de la función: 1.3347095332001904
Iteración: 5, Valor de la función: 1.309320024922995
Iteración: 6, Valor de la función: 1.309320024922995
Iteración: 7, Valor de la función: 1.2334185880740571
Iteración: 8, Valor de la función: 1.2334185880740571
Iteración: 9, Valor de la función: 1.1718628762061694
Iteración: 10, Valor de la función: 1.1718628762061694
Iteración: 11, Valor de la función: 1.1718628762061694
Iteración: 12, Valor de la función: 1.1718628762061694
Iteración: 13, Valor de la función: 1.1718628762061694
Iteración: 14, Valor de la función: 1.1483132903821045
Iteración: 15, Valor de la función: 1.1483132903821045
Iteración: 16, Valor de la función: 1.0891711100840848
Iteración: 17, Valor de la función: 1.0891711100840848
Iteración: 18, Valor de la función: 1.0891711100840848
Iteración: 19, Valor de la función: 1.0891711100840848
Iteración: 20, Valor de la función: 1.0891711100840848
Iteración: 21, Valor de la función: 1.0891711100840848
Iteración: 22, Valor de la función: 1.0843785948239864
Iteración: 23, Valor de la función: 1.0843785948239864
Iteración: 24, Valor de la función: 1.0843785948239864
Iteración: 25, Valor de la función: 1.0843785948239864
Iteración: 26, Valor de la función: 1.0843785948239864
Iteración: 27, Valor de la función: 1.0725089697717052
Iteración: 28, Valor de la función: 1.0725089697717052
Iteración: 29, Valor de la función: 1.0550680336408629
Iteración: 30, Valor de la función: 1.0550680336408629
Iteración: 31, Valor de la función: 1.0550680336408629
Iteración: 32, Valor de la función: 1.0533262275056148
Iteración: 33, Valor de la función: 1.0533262275056148
Iteración: 34, Valor de la función: 1.0463146219324093
Iteración: 35, Valor de la función: 1.0463146219324093
Iteración: 36, Valor de la función: 1.0371412651640202
Iteración: 37, Valor de la función: 1.0371412651640202
Iteración: 38, Valor de la función: 1.0371412651640202
Iteración: 39, Valor de la función: 1.0357148189700303
```

```
Iteración: 40, Valor de la función: 1.0357148189700303
Iteración: 41, Valor de la función: 1.0357148189700303
Iteración: 42, Valor de la función: 1.0357148189700303
Iteración: 43, Valor de la función: 1.0357148189700303
Iteración: 44, Valor de la función: 1.028051695295188
Iteración: 45, Valor de la función: 1.0197092577375504
Iteración: 46, Valor de la función: 1.0197092577375504
Iteración: 47, Valor de la función: 1.0197092577375504
Iteración: 48, Valor de la función: 1.0029415252797045
Iteración: 49, Valor de la función: 1.0029415252797045
Iteración: 50, Valor de la función: 1.0029415252797045
Iteración: 51, Valor de la función: 1.0029415252797045
Iteración: 52, Valor de la función: 1.0029415252797045
Iteración: 53, Valor de la función: 1.0029415252797045
Iteración: 54, Valor de la función: 1.0029415252797045
Iteración: 55, Valor de la función: 0.996216191006746
Iteración: 56, Valor de la función: 0.996216191006746
Iteración: 57, Valor de la función: 0.996216191006746
Iteración: 58, Valor de la función: 0.996216191006746
Iteración: 59, Valor de la función: 0.996216191006746
Iteración: 60, Valor de la función: 0.996216191006746
Iteración: 61, Valor de la función: 0.996216191006746
Iteración: 62, Valor de la función: 0.996216191006746
Iteración: 63, Valor de la función: 0.9943617841120955
Iteración: 64, Valor de la función: 0.9943617841120955
Iteración: 65, Valor de la función: 0.9895466771435455
Iteración: 66, Valor de la función: 0.9895466771435455
Iteración: 67, Valor de la función: 0.9895466771435455
Iteración: 68, Valor de la función: 0.9895466771435455
Iteración: 69, Valor de la función: 0.9862884563319018
Iteración: 70, Valor de la función: 0.9862884563319018
Iteración: 71, Valor de la función: 0.9862884563319018
Iteración: 72, Valor de la función: 0.9862884563319018
Iteración: 73, Valor de la función: 0.9830935231840174
Iteración: 74, Valor de la función: 0.9830935231840174
Iteración: 75, Valor de la función: 0.9830935231840174
Iteración: 76, Valor de la función: 0.9830935231840174
Iteración: 77, Valor de la función: 0.9830935231840174
Iteración: 78, Valor de la función: 0.9830935231840174
Iteración: 79, Valor de la función: 0.9830935231840174
Iteración: 80, Valor de la función: 0.9830935231840174
Iteración: 81, Valor de la función: 0.9830935231840174
Iteración: 82, Valor de la función: 0.9830935231840174
Iteración: 83, Valor de la función: 0.9830935231840174
Iteración: 84, Valor de la función: 0.9830935231840174
Iteración: 85, Valor de la función: 0.9788484553289353
Iteración: 86, Valor de la función: 0.9788484553289353
Iteración: 87, Valor de la función: 0.9788484553289353
```

```
Iteración: 88, Valor de la función: 0.9788484553289353
Iteración: 89, Valor de la función: 0.9788484553289353
Iteración: 90, Valor de la función: 0.9788484553289353
Iteración: 91, Valor de la función: 0.9788484553289353
Iteración: 92, Valor de la función: 0.9788484553289353
Iteración: 93, Valor de la función: 0.9788484553289353
Iteración: 94, Valor de la función: 0.9772067761623996
Iteración: 95, Valor de la función: 0.9772067761623996
Iteración: 96, Valor de la función: 0.9772067761623996
Iteración: 97, Valor de la función: 0.9772067761623996
Iteración: 98, Valor de la función: 0.9772067761623996
Iteración: 99, Valor de la función: 0.9772067761623996
Iteración: 100, Valor de la función: 0.9772067761623996
Iteración: 101, Valor de la función: 0.9772067761623996
Iteración: 102, Valor de la función: 0.9772067761623996
Iteración: 103, Valor de la función: 0.9772067761623996
Iteración: 104, Valor de la función: 0.9772067761623996
Iteración: 105, Valor de la función: 0.9740893604191376
Iteración: 106, Valor de la función: 0.9740893604191376
Iteración: 107, Valor de la función: 0.9738785318048803
Iteración: 108, Valor de la función: 0.9738785318048803
Iteración: 109, Valor de la función: 0.9738785318048803
Iteración: 110, Valor de la función: 0.9738785318048803
Iteración: 111, Valor de la función: 0.9731207040588491
Iteración: 112, Valor de la función: 0.9715544416188513
Iteración: 113, Valor de la función: 0.9715544416188513
Iteración: 114, Valor de la función: 0.9715544416188513
Iteración: 115, Valor de la función: 0.9715544416188513
Iteración: 116, Valor de la función: 0.9715544416188513
Iteración: 117, Valor de la función: 0.9715544416188513
Iteración: 118, Valor de la función: 0.9710472917784978
Iteración: 119, Valor de la función: 0.9710472917784978
Iteración: 120, Valor de la función: 0.9706824765915988
Iteración: 121, Valor de la función: 0.9675107994482516
Iteración: 122, Valor de la función: 0.9675107994482516
Iteración: 123, Valor de la función: 0.9675107994482516
Iteración: 124, Valor de la función: 0.9675107994482516
Iteración: 125, Valor de la función: 0.9675107994482516
Iteración: 126, Valor de la función: 0.9650150220039492
Iteración: 127, Valor de la función: 0.9650150220039492
Iteración: 128, Valor de la función: 0.9650150220039492
Iteración: 129, Valor de la función: 0.9650150220039492
Iteración: 130, Valor de la función: 0.9650150220039492
Iteración: 131, Valor de la función: 0.9650150220039492
Iteración: 132, Valor de la función: 0.9650150220039492
Iteración: 133, Valor de la función: 0.9650150220039492
Iteración: 134, Valor de la función: 0.9650150220039492
Iteración: 135, Valor de la función: 0.9650150220039492
```

```
Iteración: 136, Valor de la función: 0.9650150220039492
Iteración: 137, Valor de la función: 0.9650150220039492
Iteración: 138, Valor de la función: 0.9650150220039492
Iteración: 139, Valor de la función: 0.9650150220039492
Iteración: 140, Valor de la función: 0.9650150220039492
Iteración: 141, Valor de la función: 0.9650150220039492
Iteración: 142, Valor de la función: 0.9650150220039492
Iteración: 143, Valor de la función: 0.9650150220039492
Iteración: 144, Valor de la función: 0.9650150220039492
Iteración: 145, Valor de la función: 0.9643380231957376
Iteración: 146, Valor de la función: 0.9643380231957376
Iteración: 147, Valor de la función: 0.9643380231957376
Iteración: 148, Valor de la función: 0.9643380231957376
Iteración: 149, Valor de la función: 0.9643380231957376
Iteración: 150, Valor de la función: 0.9643380231957376
Iteración: 151, Valor de la función: 0.9643380231957376
Iteración: 152, Valor de la función: 0.9616886520165837
Iteración: 153, Valor de la función: 0.9616886520165837
Iteración: 154, Valor de la función: 0.9616886520165837
Iteración: 155, Valor de la función: 0.9616886520165837
Iteración: 156, Valor de la función: 0.9616886520165837
Iteración: 157, Valor de la función: 0.9616886520165837
Iteración: 158, Valor de la función: 0.9616886520165837
Iteración: 159, Valor de la función: 0.9616886520165837
Iteración: 160, Valor de la función: 0.9616886520165837
Iteración: 161, Valor de la función: 0.9616886520165837
Iteración: 162, Valor de la función: 0.9616886520165837
Iteración: 163, Valor de la función: 0.9616886520165837
Iteración: 164, Valor de la función: 0.9616886520165837
Iteración: 165, Valor de la función: 0.9616886520165837
Iteración: 166, Valor de la función: 0.9616886520165837
Iteración: 167, Valor de la función: 0.9616886520165837
Iteración: 168, Valor de la función: 0.9605227484730952
Iteración: 169, Valor de la función: 0.9605227484730952
Iteración: 170, Valor de la función: 0.9605227484730952
Iteración: 171, Valor de la función: 0.9605227484730952
Iteración: 172, Valor de la función: 0.9605227484730952
Iteración: 173, Valor de la función: 0.9605227484730952
Iteración: 174, Valor de la función: 0.9605227484730952
Iteración: 175, Valor de la función: 0.9603194074145992
Iteración: 176, Valor de la función: 0.9603194074145992
Iteración: 177, Valor de la función: 0.9603194074145992
Iteración: 178, Valor de la función: 0.9603194074145992
Iteración: 179, Valor de la función: 0.9603194074145992
Iteración: 180, Valor de la función: 0.9603194074145992
Iteración: 181, Valor de la función: 0.9603194074145992
Iteración: 182, Valor de la función: 0.9603194074145992
Iteración: 183, Valor de la función: 0.9603194074145992
```

```
Iteración: 184, Valor de la función: 0.9603194074145992
Iteración: 185, Valor de la función: 0.9603194074145992
Iteración: 186, Valor de la función: 0.9603194074145992
Iteración: 187, Valor de la función: 0.9603194074145992
Iteración: 188, Valor de la función: 0.9603194074145992
Iteración: 189, Valor de la función: 0.9603194074145992
Iteración: 190, Valor de la función: 0.9603194074145992
Iteración: 191, Valor de la función: 0.9599040679968291
Iteración: 192, Valor de la función: 0.9599040679968291
Iteración: 193, Valor de la función: 0.9599040679968291
Iteración: 194, Valor de la función: 0.9599040679968291
Iteración: 195, Valor de la función: 0.9599040679968291
Iteración: 196, Valor de la función: 0.9599040679968291
Iteración: 197, Valor de la función: 0.9599040679968291
Iteración: 198, Valor de la función: 0.9599040679968291
Iteración: 199, Valor de la función: 0.9599040679968291
Iteración: 200, Valor de la función: 0.9599040679968291
Iteración: 201, Valor de la función: 0.9599040679968291
Iteración: 202, Valor de la función: 0.9599040679968291
Iteración: 203, Valor de la función: 0.9599040679968291
Iteración: 204, Valor de la función: 0.9599040679968291
Iteración: 205, Valor de la función: 0.9599040679968291
Iteración: 206, Valor de la función: 0.9599040679968291
Iteración: 207, Valor de la función: 0.9599040679968291
Iteración: 208, Valor de la función: 0.9599040679968291
Iteración: 209, Valor de la función: 0.9599040679968291
Iteración: 210, Valor de la función: 0.9599040679968291
Iteración: 211, Valor de la función: 0.9599040679968291
Iteración: 212, Valor de la función: 0.9599040679968291
Iteración: 213, Valor de la función: 0.9599040679968291
Iteración: 214, Valor de la función: 0.9599040679968291
Iteración: 215, Valor de la función: 0.9599040679968291
Iteración: 216, Valor de la función: 0.9599040679968291
Iteración: 217, Valor de la función: 0.9599040679968291
Iteración: 218, Valor de la función: 0.9599040679968291
Iteración: 219, Valor de la función: 0.9599040679968291
Iteración: 220, Valor de la función: 0.9599040679968291
Iteración: 221, Valor de la función: 0.9599040679968291
Iteración: 222, Valor de la función: 0.9599040679968291
Iteración: 223, Valor de la función: 0.9599040679968291
Iteración: 224, Valor de la función: 0.9599040679968291
Iteración: 225, Valor de la función: 0.9599040679968291
Iteración: 226, Valor de la función: 0.9599040679968291
Iteración: 227, Valor de la función: 0.9599040679968291
Iteración: 228, Valor de la función: 0.9599040679968291
Iteración: 229, Valor de la función: 0.9587988552009312
Iteración: 230, Valor de la función: 0.9587988552009312
Iteración: 231, Valor de la función: 0.9587988552009312
```

```
Iteración: 232, Valor de la función: 0.9587988552009312
Iteración: 233, Valor de la función: 0.9587988552009312
Iteración: 234, Valor de la función: 0.9587988552009312
Iteración: 235, Valor de la función: 0.9587002918132554
Iteración: 236, Valor de la función: 0.9587002918132554
Iteración: 237, Valor de la función: 0.9587002918132554
Iteración: 238, Valor de la función: 0.9587002918132554
Iteración: 239, Valor de la función: 0.9587002918132554
Iteración: 240, Valor de la función: 0.9587002918132554
Iteración: 241, Valor de la función: 0.9587002918132554
Iteración: 242, Valor de la función: 0.9587002918132554
Iteración: 243, Valor de la función: 0.9587002918132554
Iteración: 244, Valor de la función: 0.9587002918132554
Iteración: 245, Valor de la función: 0.9587002918132554
Iteración: 246, Valor de la función: 0.9587002918132554
Iteración: 247, Valor de la función: 0.9587002918132554
Iteración: 248, Valor de la función: 0.956218042038391
Iteración: 249, Valor de la función: 0.956218042038391
Iteración: 250, Valor de la función: 0.956218042038391
Iteración: 251, Valor de la función: 0.956218042038391
Iteración: 252, Valor de la función: 0.956218042038391
Iteración: 253, Valor de la función: 0.956218042038391
Iteración: 254, Valor de la función: 0.956218042038391
Iteración: 255, Valor de la función: 0.956218042038391
Iteración: 256, Valor de la función: 0.956218042038391
Iteración: 257, Valor de la función: 0.956218042038391
Iteración: 258, Valor de la función: 0.956218042038391
Iteración: 259, Valor de la función: 0.956218042038391
Iteración: 260, Valor de la función: 0.956218042038391
Iteración: 261, Valor de la función: 0.956218042038391
Iteración: 262, Valor de la función: 0.956218042038391
Iteración: 263, Valor de la función: 0.956218042038391
Iteración: 264, Valor de la función: 0.956218042038391
Iteración: 265, Valor de la función: 0.956218042038391
Iteración: 266, Valor de la función: 0.956218042038391
Iteración: 267, Valor de la función: 0.955799933160606
Iteración: 268, Valor de la función: 0.955799933160606
Iteración: 269, Valor de la función: 0.955799933160606
Iteración: 270, Valor de la función: 0.955799933160606
Iteración: 271, Valor de la función: 0.955799933160606
Iteración: 272, Valor de la función: 0.955799933160606
Iteración: 273, Valor de la función: 0.955799933160606
Iteración: 274, Valor de la función: 0.955799933160606
Iteración: 275, Valor de la función: 0.955799933160606
Iteración: 276, Valor de la función: 0.955799933160606
Iteración: 277, Valor de la función: 0.955799933160606
Iteración: 278, Valor de la función: 0.955799933160606
Iteración: 279, Valor de la función: 0.955799933160606
```

```
Iteración: 280, Valor de la función: 0.955799933160606
Iteración: 281, Valor de la función: 0.955799933160606
Iteración: 282, Valor de la función: 0.955799933160606
Iteración: 283, Valor de la función: 0.955799933160606
Iteración: 284, Valor de la función: 0.955799933160606
Iteración: 285, Valor de la función: 0.955799933160606
Iteración: 286, Valor de la función: 0.955799933160606
Iteración: 287, Valor de la función: 0.955799933160606
Iteración: 288, Valor de la función: 0.955799933160606
Iteración: 289, Valor de la función: 0.955799933160606
Iteración: 290, Valor de la función: 0.955799933160606
Iteración: 291, Valor de la función: 0.955799933160606
Iteración: 292, Valor de la función: 0.955799933160606
Iteración: 293, Valor de la función: 0.955799933160606
Iteración: 294, Valor de la función: 0.955799933160606
Iteración: 295, Valor de la función: 0.955799933160606
Iteración: 296, Valor de la función: 0.955799933160606
Iteración: 297, Valor de la función: 0.955799933160606
Iteración: 298, Valor de la función: 0.955799933160606
Iteración: 299, Valor de la función: 0.955799933160606
Iteración: 300, Valor de la función: 0.955799933160606
Iteración: 301, Valor de la función: 0.955799933160606
Iteración: 302, Valor de la función: 0.955799933160606
Iteración: 303, Valor de la función: 0.955799933160606
Iteración: 304, Valor de la función: 0.955799933160606
Iteración: 305, Valor de la función: 0.955799933160606
Iteración: 306, Valor de la función: 0.955799933160606
Iteración: 307, Valor de la función: 0.955799933160606
Iteración: 308, Valor de la función: 0.955799933160606
Iteración: 309, Valor de la función: 0.955799933160606
Iteración: 310, Valor de la función: 0.955799933160606
Iteración: 311, Valor de la función: 0.955799933160606
Iteración: 312, Valor de la función: 0.955799933160606
Iteración: 313, Valor de la función: 0.955799933160606
Iteración: 314, Valor de la función: 0.955799933160606
Iteración: 315, Valor de la función: 0.955799933160606
Iteración: 316, Valor de la función: 0.955799933160606
Iteración: 317, Valor de la función: 0.955799933160606
Iteración: 318, Valor de la función: 0.955799933160606
Iteración: 319, Valor de la función: 0.955799933160606
Iteración: 320, Valor de la función: 0.955799933160606
Iteración: 321, Valor de la función: 0.955799933160606
Iteración: 322, Valor de la función: 0.955799933160606
Iteración: 323, Valor de la función: 0.955799933160606
Iteración: 324, Valor de la función: 0.955799933160606
Iteración: 325, Valor de la función: 0.955799933160606
Iteración: 326, Valor de la función: 0.955799933160606
Iteración: 327, Valor de la función: 0.955799933160606
```

```
Iteración: 328, Valor de la función: 0.955799933160606
Iteración: 329, Valor de la función: 0.955799933160606
Iteración: 330, Valor de la función: 0.955799933160606
Iteración: 331, Valor de la función: 0.955799933160606
Iteración: 332, Valor de la función: 0.955799933160606
Iteración: 333, Valor de la función: 0.955799933160606
Iteración: 334, Valor de la función: 0.955799933160606
Iteración: 335, Valor de la función: 0.955799933160606
Iteración: 336, Valor de la función: 0.955799933160606
Iteración: 337, Valor de la función: 0.955799933160606
Iteración: 338, Valor de la función: 0.955799933160606
Iteración: 339, Valor de la función: 0.955799933160606
Iteración: 340, Valor de la función: 0.955799933160606
Iteración: 341, Valor de la función: 0.955799933160606
Iteración: 342, Valor de la función: 0.955799933160606
Iteración: 343, Valor de la función: 0.955799933160606
Iteración: 344, Valor de la función: 0.955799933160606
Iteración: 345, Valor de la función: 0.955799933160606
Iteración: 346, Valor de la función: 0.955799933160606
Iteración: 347, Valor de la función: 0.955799933160606
Iteración: 348, Valor de la función: 0.955799933160606
Iteración: 349, Valor de la función: 0.955799933160606
Iteración: 350, Valor de la función: 0.955799933160606
Iteración: 351, Valor de la función: 0.955799933160606
Iteración: 352, Valor de la función: 0.955799933160606
Iteración: 353, Valor de la función: 0.955799933160606
Iteración: 354, Valor de la función: 0.955799933160606
Iteración: 355, Valor de la función: 0.955799933160606
Iteración: 356, Valor de la función: 0.955799933160606
Iteración: 357, Valor de la función: 0.955799933160606
Iteración: 358, Valor de la función: 0.955799933160606
Iteración: 359, Valor de la función: 0.955799933160606
Iteración: 360, Valor de la función: 0.955799933160606
Iteración: 361, Valor de la función: 0.955799933160606
Iteración: 362, Valor de la función: 0.955799933160606
Iteración: 363, Valor de la función: 0.955799933160606
Iteración: 364, Valor de la función: 0.955799933160606
Iteración: 365, Valor de la función: 0.955799933160606
Iteración: 366, Valor de la función: 0.955799933160606
Iteración: 367, Valor de la función: 0.955799933160606
Iteración: 368, Valor de la función: 0.955799933160606
Iteración: 369, Valor de la función: 0.955799933160606
Iteración: 370, Valor de la función: 0.955799933160606
Iteración: 371, Valor de la función: 0.955799933160606
Iteración: 372, Valor de la función: 0.955799933160606
Iteración: 373, Valor de la función: 0.955799933160606
Iteración: 374, Valor de la función: 0.9556536144954033
Iteración: 375, Valor de la función: 0.9556536144954033
```

```
Iteración: 376, Valor de la función: 0.9556536144954033
Iteración: 377, Valor de la función: 0.9556536144954033
Iteración: 378, Valor de la función: 0.9556536144954033
Iteración: 379, Valor de la función: 0.9556536144954033
Iteración: 380, Valor de la función: 0.9556536144954033
Iteración: 381, Valor de la función: 0.9553980770369657
Iteración: 382, Valor de la función: 0.9553980770369657
Iteración: 383, Valor de la función: 0.9553980770369657
Iteración: 384, Valor de la función: 0.9553980770369657
Iteración: 385, Valor de la función: 0.9553980770369657
Iteración: 386, Valor de la función: 0.9553980770369657
Iteración: 387, Valor de la función: 0.9553980770369657
Iteración: 388, Valor de la función: 0.9553980770369657
Iteración: 389, Valor de la función: 0.9553980770369657
Iteración: 390, Valor de la función: 0.9553980770369657
Iteración: 391, Valor de la función: 0.9553980770369657
Iteración: 392, Valor de la función: 0.9553980770369657
Iteración: 393, Valor de la función: 0.9553980770369657
Iteración: 394, Valor de la función: 0.9553980770369657
Iteración: 395, Valor de la función: 0.9553391877145138
Iteración: 396, Valor de la función: 0.9553391877145138
Iteración: 397, Valor de la función: 0.9553391877145138
Iteración: 398, Valor de la función: 0.9553391877145138
Iteración: 399, Valor de la función: 0.9553391877145138
Iteración: 400, Valor de la función: 0.9553391877145138
Iteración: 401, Valor de la función: 0.9553391877145138
Iteración: 402, Valor de la función: 0.9553391877145138
Iteración: 403, Valor de la función: 0.9550789365044831
Iteración: 404, Valor de la función: 0.9550789365044831
Iteración: 405, Valor de la función: 0.9550789365044831
Iteración: 406, Valor de la función: 0.9550789365044831
Iteración: 407, Valor de la función: 0.9550789365044831
Iteración: 408, Valor de la función: 0.9550789365044831
Iteración: 409, Valor de la función: 0.9550789365044831
Iteración: 410, Valor de la función: 0.9550789365044831
Iteración: 411, Valor de la función: 0.9550789365044831
Iteración: 412, Valor de la función: 0.9550789365044831
Iteración: 413, Valor de la función: 0.9550789365044831
Iteración: 414, Valor de la función: 0.9550789365044831
Iteración: 415, Valor de la función: 0.9550789365044831
Iteración: 416, Valor de la función: 0.9550789365044831
Iteración: 417, Valor de la función: 0.9550789365044831
Iteración: 418, Valor de la función: 0.9550789365044831
Iteración: 419, Valor de la función: 0.9550789365044831
Iteración: 420, Valor de la función: 0.9550789365044831
Iteración: 421, Valor de la función: 0.9550789365044831
Iteración: 422, Valor de la función: 0.9550789365044831
Iteración: 423, Valor de la función: 0.9550789365044831
```

```
Iteración: 424, Valor de la función: 0.9550789365044831
Iteración: 425, Valor de la función: 0.9550789365044831
Iteración: 426, Valor de la función: 0.9550789365044831
Iteración: 427, Valor de la función: 0.9550789365044831
Iteración: 428, Valor de la función: 0.9550789365044831
Iteración: 429, Valor de la función: 0.9550789365044831
Iteración: 430, Valor de la función: 0.9550789365044831
Iteración: 431, Valor de la función: 0.9550789365044831
Iteración: 432, Valor de la función: 0.9550789365044831
Iteración: 433, Valor de la función: 0.9550789365044831
Iteración: 434, Valor de la función: 0.9550789365044831
Iteración: 435, Valor de la función: 0.9550789365044831
Iteración: 436, Valor de la función: 0.9550789365044831
Iteración: 437, Valor de la función: 0.9550789365044831
Iteración: 438, Valor de la función: 0.9550789365044831
Iteración: 439, Valor de la función: 0.9550789365044831
Iteración: 440, Valor de la función: 0.9550789365044831
Iteración: 441, Valor de la función: 0.9550789365044831
Iteración: 442, Valor de la función: 0.9550789365044831
Iteración: 443, Valor de la función: 0.9550789365044831
Iteración: 444, Valor de la función: 0.9550789365044831
Iteración: 445, Valor de la función: 0.9550789365044831
Iteración: 446, Valor de la función: 0.9550789365044831
Iteración: 447, Valor de la función: 0.9550789365044831
Iteración: 448, Valor de la función: 0.9550789365044831
Iteración: 449, Valor de la función: 0.9550789365044831
Iteración: 450, Valor de la función: 0.9550789365044831
Iteración: 451, Valor de la función: 0.9550789365044831
```

[256]:

```
             message: Optimization terminated successfully.
             success: True
                 fun: 0.9537486915239393
                   x: [ 9.696e-01  9.428e-01 ...  9.408e-01  9.684e-01]
                 nit: 451
                nfev: 651431
          population: [[ 9.735e-01  9.415e-01 ...  9.437e-01  9.637e-01]
                       [ 9.593e-01  8.938e-01 ...  8.869e-01  9.354e-01]
                       ...
                       [ 9.614e-01  9.309e-01 ...  9.255e-01  9.606e-01]
                       [ 9.308e-01  9.079e-01 ...  8.804e-01  9.163e-01]]
 population_energies: [ 9.537e-01  9.868e-01 ...  9.582e-01  9.949e-01]
                 jac: [ 6.795e-06  7.105e-07 ...  5.640e-06  6.550e-06]
```

[257]:

```python
tol_vol = 1e-2 # tolerancia con la restricción
cons_de = NonlinearConstraint(vol_func, -tol_vol, tol_vol)

sol_de_cons = differential_evolution(
```

```python
    area_func,
    bounds=bounds,
    constraints=cons_de,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
    tol=0.01,
    # disp=True,
    callback=logger
)
Famp_de_cons = np.concatenate(([F0], sol_de_cons.x, [F1]))
sol_de_cons
```

```
Iteración: 1, Valor de la función: 2.3383883018869085
Iteración: 2, Valor de la función: 2.3383883018869085
Iteración: 3, Valor de la función: 1.7134669066002162
Iteración: 4, Valor de la función: 1.7134669066002162
Iteración: 5, Valor de la función: 1.7134669066002162
Iteración: 6, Valor de la función: 1.7134669066002162
Iteración: 7, Valor de la función: 1.5141695578186096
Iteración: 8, Valor de la función: 1.3256617485565831
Iteración: 9, Valor de la función: 1.3256617485565831
Iteración: 10, Valor de la función: 1.3256617485565831
Iteración: 11, Valor de la función: 1.3061569495114662
Iteración: 12, Valor de la función: 1.3061569495114662
Iteración: 13, Valor de la función: 1.3061569495114662
Iteración: 14, Valor de la función: 1.3061569495114662
Iteración: 15, Valor de la función: 1.1847453244245074
Iteración: 16, Valor de la función: 1.1847453244245074
Iteración: 17, Valor de la función: 1.1847453244245074
Iteración: 18, Valor de la función: 1.1847453244245074
Iteración: 19, Valor de la función: 1.1847453244245074
Iteración: 20, Valor de la función: 1.1515010130867376
Iteración: 21, Valor de la función: 1.1515010130867376
Iteración: 22, Valor de la función: 1.1515010130867376
Iteración: 23, Valor de la función: 1.1515010130867376
Iteración: 24, Valor de la función: 1.1515010130867376
Iteración: 25, Valor de la función: 1.1515010130867376
Iteración: 26, Valor de la función: 1.1515010130867376
Iteración: 27, Valor de la función: 1.1515010130867376
Iteración: 28, Valor de la función: 1.1073244738552364
Iteración: 29, Valor de la función: 1.1073244738552364
Iteración: 30, Valor de la función: 1.1073244738552364
Iteración: 31, Valor de la función: 1.097584879153907
Iteración: 32, Valor de la función: 1.0720189695276845
Iteración: 33, Valor de la función: 1.0720189695276845
```

```
Iteración: 34, Valor de la función: 1.0720189695276845
Iteración: 35, Valor de la función: 1.0720189695276845
Iteración: 36, Valor de la función: 1.0720189695276845
Iteración: 37, Valor de la función: 1.0570393696062186
Iteración: 38, Valor de la función: 1.0284447732105422
Iteración: 39, Valor de la función: 1.0284447732105422
Iteración: 40, Valor de la función: 1.0284447732105422
Iteración: 41, Valor de la función: 1.0198007892031435
Iteración: 42, Valor de la función: 1.0198007892031435
Iteración: 43, Valor de la función: 1.0198007892031435
Iteración: 44, Valor de la función: 1.0198007892031435
Iteración: 45, Valor de la función: 1.0198007892031435
Iteración: 46, Valor de la función: 1.0147426563503554
Iteración: 47, Valor de la función: 1.0147426563503554
Iteración: 48, Valor de la función: 1.0147426563503554
Iteración: 49, Valor de la función: 1.0074692930992135
Iteración: 50, Valor de la función: 1.007214554150368
Iteración: 51, Valor de la función: 1.0046304350524438
Iteración: 52, Valor de la función: 1.0046304350524438
Iteración: 53, Valor de la función: 1.0033545195565319
Iteración: 54, Valor de la función: 1.0033545195565319
Iteración: 55, Valor de la función: 1.0033545195565319
Iteración: 56, Valor de la función: 1.0033545195565319
Iteración: 57, Valor de la función: 1.0002738078711866
Iteración: 58, Valor de la función: 1.0002738078711866
Iteración: 59, Valor de la función: 1.0002738078711866
Iteración: 60, Valor de la función: 1.0002738078711866
Iteración: 61, Valor de la función: 0.998402243714739
Iteración: 62, Valor de la función: 0.998402243714739
Iteración: 63, Valor de la función: 0.998402243714739
Iteración: 64, Valor de la función: 0.9799474887038457
Iteración: 65, Valor de la función: 0.9799474887038457
Iteración: 66, Valor de la función: 0.9799474887038457
Iteración: 67, Valor de la función: 0.9799474887038457
Iteración: 68, Valor de la función: 0.9799474887038457
Iteración: 69, Valor de la función: 0.9757757723246515
Iteración: 70, Valor de la función: 0.9757563209823805
Iteración: 71, Valor de la función: 0.9736076858790936
Iteración: 72, Valor de la función: 0.9736076858790936
Iteración: 73, Valor de la función: 0.9675970650386152
Iteración: 74, Valor de la función: 0.9675970650386152
Iteración: 75, Valor de la función: 0.9675970650386152
Iteración: 76, Valor de la función: 0.9675970650386152
Iteración: 77, Valor de la función: 0.9629708998343628
Iteración: 78, Valor de la función: 0.9629708998343628
Iteración: 79, Valor de la función: 0.9629708998343628
Iteración: 80, Valor de la función: 0.9629708998343628
Iteración: 81, Valor de la función: 0.9629708998343628
```

```
Iteración: 82, Valor de la función: 0.9629708998343628
Iteración: 83, Valor de la función: 0.9629708998343628
Iteración: 84, Valor de la función: 0.958509729880962
Iteración: 85, Valor de la función: 0.958509729880962
Iteración: 86, Valor de la función: 0.958509729880962
```

[257]:

```
             message: Optimization terminated successfully.
             success: True
                 fun: 0.9537486906319604
                   x: [ 9.696e-01  9.428e-01 ...  9.408e-01  9.684e-01]
                 nit: 86
                nfev: 51775
          population: [[ 9.735e-01  9.431e-01 ...  9.384e-01  9.701e-01]
                       [ 9.783e-01  9.407e-01 ...  9.554e-01  9.729e-01]
                       ...
                       [ 9.717e-01  9.586e-01 ...  9.463e-01  9.653e-01]
                       [ 9.724e-01  9.331e-01 ...  9.412e-01  9.797e-01]]
 population_energies: [ 9.537e-01  9.870e-01 ...  9.773e-01  9.808e-01]
              constr: [array([ 0.000e+00])]
    constr_violation: 0.0
               maxcv: 0.0
                 jac: [array([[ 1.021e-01,  9.924e-02, ...,  9.903e-02,
                               1.019e-01]]), array([[ 1.000e+00,  0.000e+00,
...,  0.000e+00,
                               0.000e+00],
                             [ 0.000e+00,  1.000e+00, ...,  0.000e+00,
                               0.000e+00],
                             ...,
                             [ 0.000e+00,  0.000e+00, ...,  1.000e+00,
                               0.000e+00],
                             [ 0.000e+00,  0.000e+00, ...,  0.000e+00,
                               1.000e+00]])]
```

1.5 Problema sin restricciones, con soportes de diferente tamaño

A continuación, se considera el caso en el que ambos soportes no son iguales, si no que las condiciones de contorno impuestas anteriormente se sustituyen por las siguientes:

$F(0) = F_0 - \varepsilon$

$F(1) = F_0 + \varepsilon$

Con esto, el problema pierde su simetría, por lo que se presume que aumentará la potencia de cálculo requerida para encontrar una solución. A continuación, se van a discutir los efectos de este cambio en las soluciones del problema.

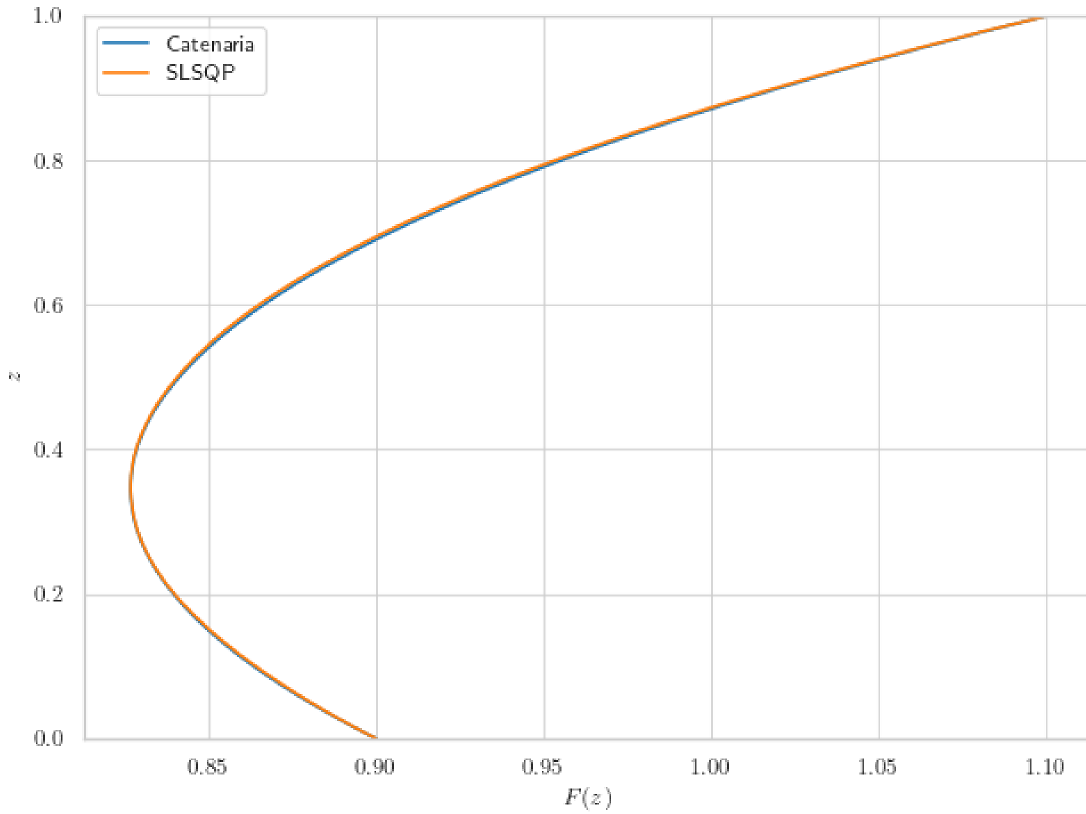
1.5.1 Minimización con un método basado en gradiente.

[8]:
```
F_0 = 1
eps = 0.1
F0 = F_0 - eps
F1 = F_0 + eps

n = 100
z = np.linspace(0, 1, n)

F_init = np.linspace(F0, F1, n-2)

lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

[9]:
```
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cat = catenaria(z)
```

[10]:
```
sol_grad_sop = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_sop = np.concatenate(([F0], sol_grad_sop.x, [F1]))
error_sop = np.linalg.norm(Famp_sop - F_cat)/np.linalg.norm(F_cat)
print('El error para la solucion estándar es:', error_sop)

plt.plot(F_cat, z, label='Catenaria')
plt.plot(Famp_sop, z, label='SLSQP')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.title('Curva optimizada con soportes de diferente tamaño, para $ \epsilon␣
↪= {}$'.format(eps))
plt.savefig('Figuras/sol_eps_SLSQP.pdf', format='pdf')
plt.show()
```

```
El error para la solucion estándar es: 0.0011578910106290717
```



Influencia de la diferencia entre los soportes, ε

[395]:
```
F_0 = 1
paso = 0.01
F0 = F_0 - eps
F1 = F_0 + eps

n = 100
z = np.linspace(0, 1, n)

F_init = np.linspace(F0, F1, n-2)

lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cat = catenaria(z)
```

```
C:\Users\ismag\AppData\Local\Temp\ipykernel_19584\1556695652.py:15:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
```

```
  a_sol, k_sol = fsolve(params_catenaria, initial_guess)
```

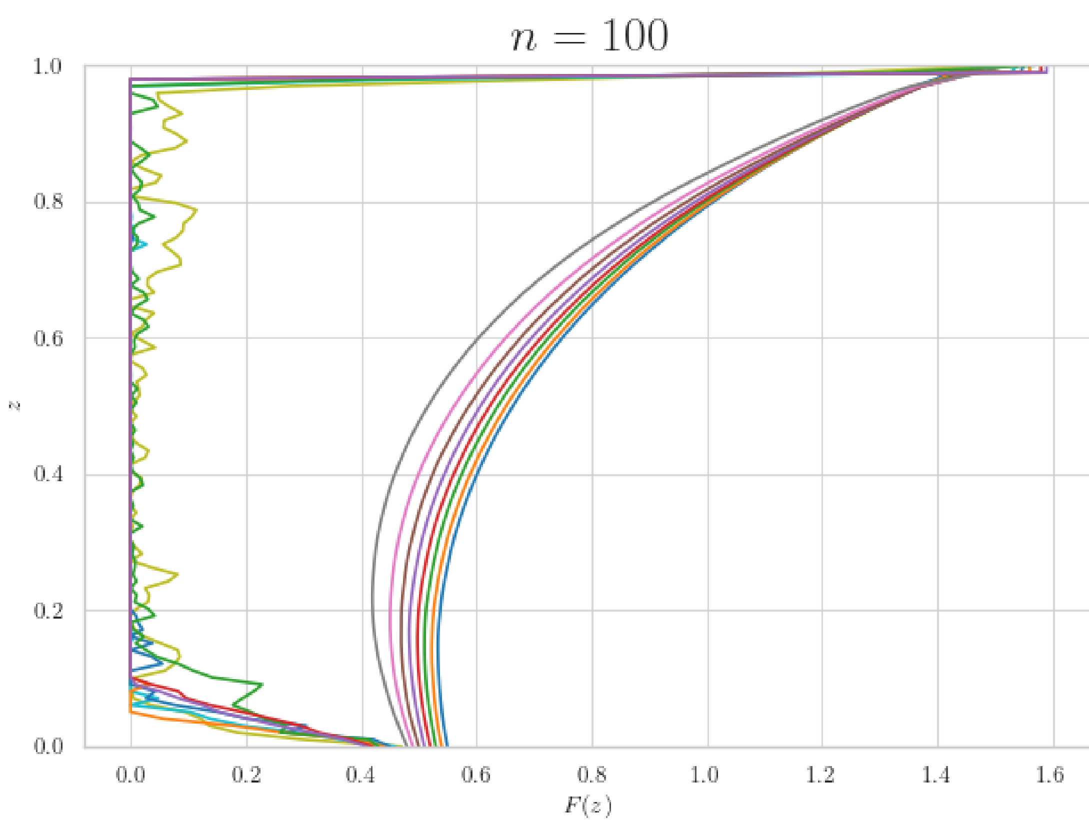
[396]:
```python
sol_sop_bucle = []
Famp_sop_bucle = []
error_sop_bucle = []
iter = 15
for i in range(45, 45+iter):
    eps = paso*i
    F0 = F_0 - eps
    F1 = F_0 + eps

    a_sol, k_sol = fsolve(params_catenaria, initial_guess)
    F_cat = catenaria(z)

    F_init = np.linspace(F0, F1, n-2)
    sol_aux =  minimize(area_func, F_init, bounds=bounds, method='SLSQP')
    Famp_aux = np.concatenate(([F0], sol_aux.x, [F1]))
    Famp_sop_bucle += [Famp_aux]
    error_aux = np.linalg.norm(Famp_aux - F_cat)/np.linalg.norm(F_cat)
    error_sop_bucle.append([eps, error_aux])

    print('El error eps = ', eps, ' es:', error_aux)
```

```
El error eps =  0.45  es: 0.021898150704899005
El error eps =  0.46  es: 0.02446139978305074
El error eps =  0.47000000000000003  es: 0.0295740951228807
El error eps =  0.48  es: 0.03356551515997064
El error eps =  0.49  es: 0.040097856157851594
El error eps =  0.5  es: 0.04954948755814675
El error eps =  0.51  es: 0.0638339774056542
El error eps =  0.52  es: 0.09680091132359257
El error eps =  0.53  es: 0.8943925772930262
El error eps =  0.54  es: 0.9413397814108504
El error eps =  0.55  es: 0.9479596919498794
El error eps =  0.56  es: 0.9499528664014313

C:\Users\ismag\AppData\Local\Temp\ipykernel_19584\519168017.py:10:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  a_sol, k_sol = fsolve(params_catenaria, initial_guess)

El error eps =  0.5700000000000001  es: 0.9144197638428311
El error eps =  0.58  es: 0.9405949366533307
El error eps =  0.59  es: 0.9415685901545178
```

[397]:
```python
error_sop_bucle = np.array(error_sop_bucle)
# error_sop_bucle
```

[398]:
```
for i in range(iter):
    plt.plot(Famp_sop_bucle[i], z, label='$\epsilon = {:.2f}$'.format(paso*i))

# plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.title('$n = {}$'.format(n), fontsize=20)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.savefig('Figuras/sol_var_epsn20.pdf', format='pdf')
# plt.savefig('Figuras/sol_var_epsn100.pdf', format='pdf')
# plt.savefig('Figuras/sol_var_epsn100_cont.pdf', format='pdf')
# plt.savefig('Figuras/Presentación/sol_var_epsn20.pdf', format='pdf')
# plt.savefig('Figuras/Presentación/sol_var_epsn100.pdf', format='pdf')
plt.savefig('Figuras/Presentación/sol_var_epsn100_cont.pdf', format='pdf')
plt.show()
```

```
<>:2: SyntaxWarning: invalid escape sequence '\e'
<>:2: SyntaxWarning: invalid escape sequence '\e'
C:\Users\ismag\AppData\Local\Temp\ipykernel_19584\3979177710.py:2:
SyntaxWarning: invalid escape sequence '\e'
  plt.plot(Famp_sop_bucle[i], z, label='$\epsilon = {:.2f}$'.format(paso*i))
```



Influencia de la distribución de puntos. En este caso, se va a considerar el efecto de considerar diferentes distribuciones de puntos en las que dividir el intervalo del problema.

Una vez más, se consideran el resto de parámetros idénticos al primer caso de estudio, salvo por el número de puntos n, que se va a modificar a fin de apreciar el comportamiento de distintas distribuciones de puntos en los casos más exigentes, es decir, en los casos extremos de n muy grande o muy pequeño. El caso de n intermedio se omite por presentar soluciones casi idénticas en todos los casos.

[ ]:
```
F_0 = 1
eps = 0.1
F0 = F_0 - eps
F1 = F_0 + eps
```

Caso de n muy pequeño.

[ ]:
```
n = 20
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

# planteamos la función que calcula el área tomando delta_z como un array␣
↪definido anteriormente
def area_distrib(F):
    Famp = np.append(np.array([F0]), np.append(F, F1))
    n = np.size(Famp)

    integral = 0
    for i in range(n-1):
        # calcular la derivada
        F_der = (Famp[i+1] - Famp[i]) / delta_z[i] # esquema adelantado para el␣
↪cálculo de la derivada
        # calcular el valor de la integral
        integrando = Famp[i] * np.sqrt(1 + F_der**2)
        integral += integrando * delta_z[i]

    return integral
```

- Distribución de puntos **equiespaciada**

[ ]:
```
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_func, F_init, method='SLSQP', bounds=bounds)
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
```

```
F_cateq = catenaria(zeq)
error_eq = np.linalg.norm(Famp_eq - F_cateq)/np.linalg.norm(F_cateq)
print('El error para distribución equiespaciada es:', error_eq)
```

```
El error para distribución equiespaciada es: 0.0035191060883840814
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**: $z_i = \cos\left(\frac{\frac{\pi}{2}+\pi i}{n+1}\right), \quad i = 0, \ldots, n$

[ ]:
```
def chebyshev_zeros(a, b, n):
    cheb_zeros = np.cos((2 * np.arange(1, n+1) - 1) * np.pi / (2 * n))
    mapped_zeros = 0.5 * (b - a) * (cheb_zeros + 1) + a
    sorted_zeros = np.sort(mapped_zeros) # porque los calcula en orden␣
 ↪decreciente
    return sorted_zeros

zchebz = chebyshev_zeros(0, 1, n)
delta_z = np.diff(zchebz)
sol_gradchebz = minimize(area_func, F_init, method='SLSQP', bounds=bounds)
Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebz = catenaria(zchebz)
error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)/np.linalg.norm(F_catchebz)
print('El error para los nodos de Chebyshev es:', error_chebz)
```

```
El error para los nodos de Chebyshev es: 0.03394472870761714
```

- Distribución de puntos según un esquema de **extremos de Chebyshev**: $z_i = \cos\left(\frac{\pi i}{n}\right), \quad i = 0, \ldots, n$

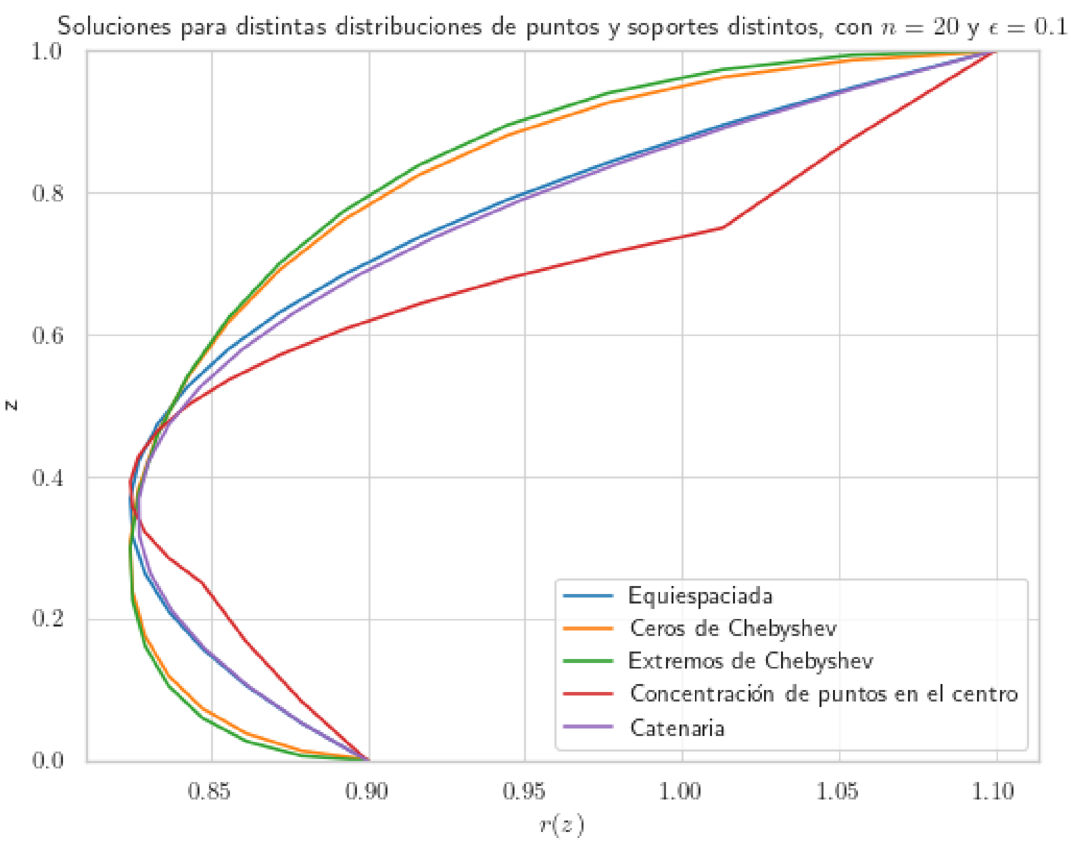
[ ]:
```
def chebyshev_extremes(a, b, n):
    cheb_extremes = np.cos(np.arange(n) * np.pi / (n - 1))
    mapped_extremes = 0.5 * (b - a) * (cheb_extremes + 1) + a
    sorted_extremes = np.sort(mapped_extremes)
    return sorted_extremes

zchebe = chebyshev_extremes(0, 1, n)
delta_z = np.diff(zchebe)
sol_gradchebe = minimize(area_func, F_init, method='SLSQP', bounds=bounds)
Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebe = catenaria(zchebe)
error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)/np.linalg.norm(F_catchebe)
print('El error para los extremos de Chebyshev es:', error_chebe)
```

```
El error para los extremos de Chebyshev es: 0.03938968250050415
```

[ ]:
```
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
```

```
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
plt.plot(F_cateq, zeq, label='Catenaria')
plt.ylabel('z')
plt.xlabel('$r(z)$')
plt.ylim(0, 1)
# plt.xlim(0, 1.3)
plt.title('Soluciones para distintas distribuciones de puntos y soportes␣
↪distintos, con $n = {}$ y $\epsilon = {}$'.format(n, eps))
plt.legend()
plt.grid(True)
plt.show()
```

```
<>:10: SyntaxWarning: invalid escape sequence '\e'
<>:10: SyntaxWarning: invalid escape sequence '\e'
C:\Users\ismag\AppData\Local\Temp\ipykernel_6660\1885633864.py:10:
SyntaxWarning: invalid escape sequence '\e'
  plt.title('Soluciones para distintas distribuciones de puntos y soportes
distintos, con $n = {}$ y $\epsilon = {}$'.format(n, eps))
```



A primera vista, todas las soluciones son extremadamente similares, por lo que se pueden considerar todas igual de válidas. Sin embargo, al ampliar y observar la solución de una forma más cercana, se aprecia que la única que cumple la restricción de $V = 1$ y además se corresponde con la solución esperada es la equiespaciada.

NO SÉ EXPLICAR POR QUÉ ESO ES ASÍ

Caso de n muy grande.

[ ]:
```
n = 100
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T
```

- Distribución de puntos **equiespaciada**

[ ]:
```
zeq = np.linspace(0, 1, n)
delta_z = np.diff(zeq)
sol_gradeq = minimize(area_func, F_init, method='SLSQP', bounds=bounds)
Famp_eq = np.concatenate(([F0], sol_gradeq.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cateq = catenaria(zeq)
error_eq = np.linalg.norm(Famp_eq - F_cateq)/np.linalg.norm(F_cateq)
print('El error para distribución equiespaciada es:', error_eq)
```

```
El error para distribución equiespaciada es: 0.001171777526170807
```

- Distribución de puntos según un esquema de **ceros de Chebyshev**: $z_i = \cos\left(\frac{\frac{\pi}{2}+\pi i}{n+1}\right), \quad i = 0, \ldots, n$

[ ]:
```
zchebz = chebyshev_zeros(0, 1, n)
delta_z = np.diff(zchebz)
sol_gradchebz = minimize(area_func, F_init, method='SLSQP', bounds=bounds)
Famp_chebz = np.concatenate(([F0], sol_gradchebz.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebz = catenaria(zchebz)
error_chebz = np.linalg.norm(Famp_chebz - F_catchebz)/np.linalg.norm(F_catchebz)
print('El error para los nodos de Chebyshev es:', error_chebz)
```

```
El error para los nodos de Chebyshev es: 0.03701329176383531
```

- Distribución de puntos según un esquema de **extremos de Chebyshev**: $z_i = \cos\left(\frac{\pi i}{n}\right), \quad i = 0, \ldots, n$

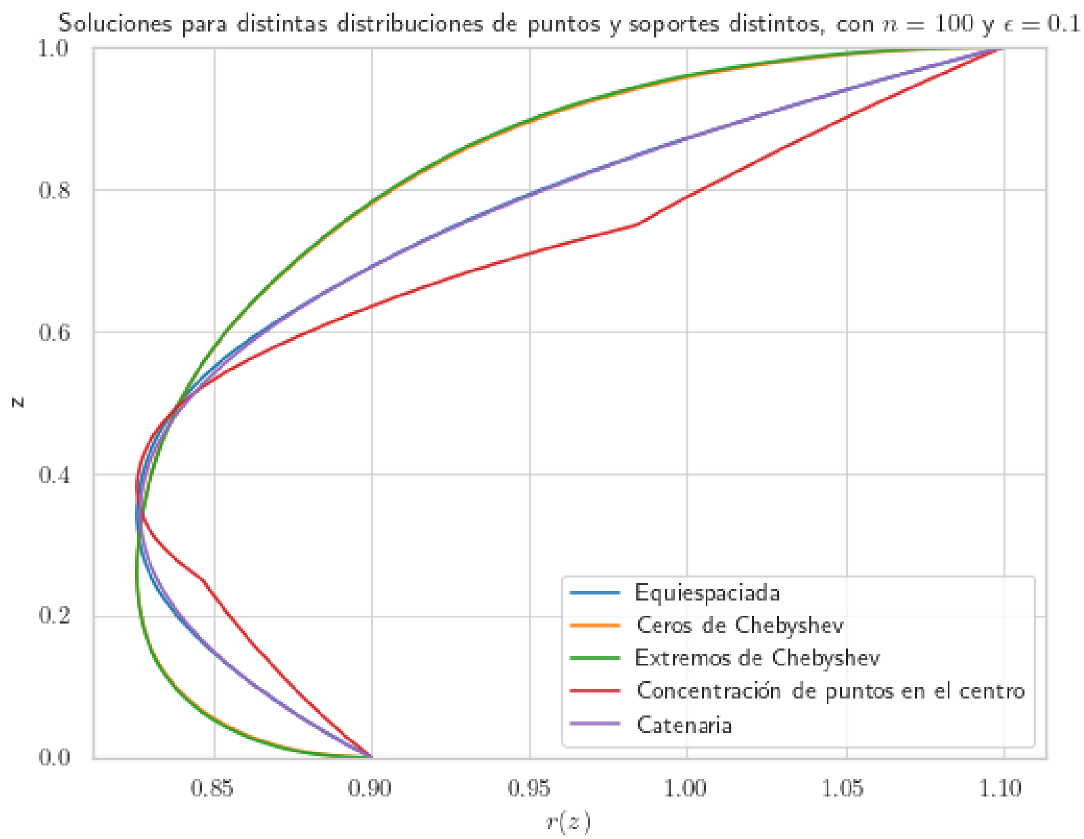
[ ]:
```
zchebe = chebyshev_extremes(0, 1, n)
delta_z = np.diff(zchebe)
sol_gradchebe = minimize(area_func, F_init, method='SLSQP', bounds=bounds)
Famp_chebe = np.concatenate(([F0], sol_gradchebe.x, [F1]))
a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_catchebe = catenaria(zchebe)
error_chebe = np.linalg.norm(Famp_chebe - F_catchebe)/np.linalg.norm(F_catchebe)
```

```
print('El error para los extremos de Chebyshev es:', error_chebe)
```

```
El error para los extremos de Chebyshev es: 0.03809766405020637
```

[ ]:
```
plt.plot(Famp_eq, zeq, label = 'Equiespaciada')
plt.plot(Famp_chebz, zchebz, label = 'Ceros de Chebyshev')
plt.plot(Famp_chebe, zchebe, label = 'Extremos de Chebyshev')
plt.plot(F_cateq, zeq, label='Catenaria')
plt.ylabel('z')
plt.xlabel('$r(z)$')
plt.ylim(0, 1)
# plt.xlim(0, 1.3)
plt.title('Soluciones para distintas distribuciones de puntos y soportes␣
 ↪distintos, con $n = {}$ y $\epsilon = {}$'.format(n, eps))
plt.legend()
plt.grid(True)
plt.show()
```

```
<>:10: SyntaxWarning: invalid escape sequence '\e'
<>:10: SyntaxWarning: invalid escape sequence '\e'
C:\Users\ismag\AppData\Local\Temp\ipykernel_6660\1885633864.py:10:
SyntaxWarning: invalid escape sequence '\e'
  plt.title('Soluciones para distintas distribuciones de puntos y soportes
distintos, con $n = {}$ y $\epsilon = {}$'.format(n, eps))
```



1.5.2 Minimización con un método heurístico (*Differential Evolution*)

[ ]:
```
F_0 = 1
eps = 0.1
F0 = F_0 - eps
F1 = F_0 + eps

n = 20
z = np.linspace(0, 1, n)

bounds = [(F_0-0.5, F_0+0.5) for _ in range(n-2)]

sol_sop = differential_evolution(
    area_func,
    bounds=bounds,
    maxiter=100000,
    popsize=4*n,
    mutation=(0.5, 1),
    recombination=0.7,
    strategy='best1bin',
```

```
    tol=0.01,
    callback=logger
)
```

```
Iteración: 1, Valor de la función: 2.574430667920992
Iteración: 2, Valor de la función: 2.313177310168412
Iteración: 3, Valor de la función: 2.0779651375199504
Iteración: 4, Valor de la función: 2.0779651375199504
Iteración: 5, Valor de la función: 1.9347969659093809
Iteración: 6, Valor de la función: 1.9347969659093809
Iteración: 7, Valor de la función: 1.9347969659093809
Iteración: 8, Valor de la función: 1.9347969659093809
Iteración: 9, Valor de la función: 1.7200944438539119
Iteración: 10, Valor de la función: 1.6319722952613793
Iteración: 11, Valor de la función: 1.6319722952613793
Iteración: 12, Valor de la función: 1.4907143130345364
Iteración: 13, Valor de la función: 1.4907143130345364
Iteración: 14, Valor de la función: 1.4907143130345364
Iteración: 15, Valor de la función: 1.2606670860280016
Iteración: 16, Valor de la función: 1.2606670860280016
Iteración: 17, Valor de la función: 1.2606670860280016
Iteración: 18, Valor de la función: 1.2243707781024042
Iteración: 19, Valor de la función: 1.2243707781024042
Iteración: 20, Valor de la función: 1.1633285661554746
Iteración: 21, Valor de la función: 1.1633285661554746
Iteración: 22, Valor de la función: 1.1355128504338254
Iteración: 23, Valor de la función: 1.1355128504338254
Iteración: 24, Valor de la función: 1.1355128504338254
Iteración: 25, Valor de la función: 1.1355128504338254
Iteración: 26, Valor de la función: 1.085412014370706
Iteración: 27, Valor de la función: 1.085412014370706
Iteración: 28, Valor de la función: 1.085412014370706
Iteración: 29, Valor de la función: 1.085412014370706
Iteración: 30, Valor de la función: 1.085412014370706
Iteración: 31, Valor de la función: 1.085412014370706
Iteración: 32, Valor de la función: 1.0352800370433035
Iteración: 33, Valor de la función: 1.0352800370433035
Iteración: 34, Valor de la función: 1.0352800370433035
Iteración: 35, Valor de la función: 1.0352800370433035
Iteración: 36, Valor de la función: 1.0352800370433035
Iteración: 37, Valor de la función: 1.0352800370433035
Iteración: 38, Valor de la función: 1.0352800370433035
Iteración: 39, Valor de la función: 1.0352800370433035
Iteración: 40, Valor de la función: 1.0344414265775057
Iteración: 41, Valor de la función: 1.0344414265775057
Iteración: 42, Valor de la función: 1.0237020388056308
Iteración: 43, Valor de la función: 1.0237020388056308
Iteración: 44, Valor de la función: 1.0237020388056308
```

```
Iteración: 45, Valor de la función: 1.0237020388056308
Iteración: 46, Valor de la función: 1.018498757439606
Iteración: 47, Valor de la función: 1.018498757439606
Iteración: 48, Valor de la función: 1.018498757439606
Iteración: 49, Valor de la función: 1.018498757439606
Iteración: 50, Valor de la función: 1.014550679681255
Iteración: 51, Valor de la función: 1.014550679681255
Iteración: 52, Valor de la función: 1.014550679681255
Iteración: 53, Valor de la función: 1.014550679681255
Iteración: 54, Valor de la función: 1.014550679681255
Iteración: 55, Valor de la función: 1.0066825512752076
Iteración: 56, Valor de la función: 1.0016672916157618
Iteración: 57, Valor de la función: 1.0012723535531742
Iteración: 58, Valor de la función: 1.0012723535531742
Iteración: 59, Valor de la función: 1.0012723535531742
Iteración: 60, Valor de la función: 1.0012723535531742
Iteración: 61, Valor de la función: 1.0012723535531742
Iteración: 62, Valor de la función: 1.0012723535531742
Iteración: 63, Valor de la función: 1.0012723535531742
Iteración: 64, Valor de la función: 1.0012723535531742
Iteración: 65, Valor de la función: 1.0012723535531742
Iteración: 66, Valor de la función: 1.0000023068354107
Iteración: 67, Valor de la función: 0.995382483164768
Iteración: 68, Valor de la función: 0.995382483164768
Iteración: 69, Valor de la función: 0.995382483164768
Iteración: 70, Valor de la función: 0.9952270826209225
Iteración: 71, Valor de la función: 0.9952270826209225
Iteración: 72, Valor de la función: 0.9952270826209225
Iteración: 73, Valor de la función: 0.9944988029115727
Iteración: 74, Valor de la función: 0.9944988029115727
Iteración: 75, Valor de la función: 0.9944988029115727
Iteración: 76, Valor de la función: 0.9944988029115727
Iteración: 77, Valor de la función: 0.9878196629264167
Iteración: 78, Valor de la función: 0.9878196629264167
Iteración: 79, Valor de la función: 0.9878196629264167
Iteración: 80, Valor de la función: 0.9878196629264167
Iteración: 81, Valor de la función: 0.9878196629264167
Iteración: 82, Valor de la función: 0.9878196629264167
Iteración: 83, Valor de la función: 0.9878196629264167
Iteración: 84, Valor de la función: 0.9878196629264167
Iteración: 85, Valor de la función: 0.9878196629264167
Iteración: 86, Valor de la función: 0.9878196629264167
Iteración: 87, Valor de la función: 0.9878196629264167
Iteración: 88, Valor de la función: 0.9878196629264167
Iteración: 89, Valor de la función: 0.9878196629264167
Iteración: 90, Valor de la función: 0.9854312819392669
Iteración: 91, Valor de la función: 0.9854312819392669
Iteración: 92, Valor de la función: 0.9854312819392669
```

```
Iteración: 93, Valor de la función: 0.9854312819392669
Iteración: 94, Valor de la función: 0.9854312819392669
Iteración: 95, Valor de la función: 0.9842097915636021
Iteración: 96, Valor de la función: 0.9842097915636021
Iteración: 97, Valor de la función: 0.9842097915636021
Iteración: 98, Valor de la función: 0.9842097915636021
Iteración: 99, Valor de la función: 0.9842097915636021
Iteración: 100, Valor de la función: 0.9842097915636021
Iteración: 101, Valor de la función: 0.9842097915636021
Iteración: 102, Valor de la función: 0.9842097915636021
Iteración: 103, Valor de la función: 0.983401919423405
Iteración: 104, Valor de la función: 0.983401919423405
Iteración: 105, Valor de la función: 0.9820198169648591
Iteración: 106, Valor de la función: 0.9820198169648591
Iteración: 107, Valor de la función: 0.9820198169648591
Iteración: 108, Valor de la función: 0.9820198169648591
Iteración: 109, Valor de la función: 0.9820198169648591
Iteración: 110, Valor de la función: 0.9820198169648591
Iteración: 111, Valor de la función: 0.9820198169648591
Iteración: 112, Valor de la función: 0.9820198169648591
Iteración: 113, Valor de la función: 0.9820198169648591
Iteración: 114, Valor de la función: 0.9820198169648591
Iteración: 115, Valor de la función: 0.9820198169648591
Iteración: 116, Valor de la función: 0.9820198169648591
Iteración: 117, Valor de la función: 0.9820198169648591
Iteración: 118, Valor de la función: 0.9820198169648591
Iteración: 119, Valor de la función: 0.9820198169648591
Iteración: 120, Valor de la función: 0.9820198169648591
Iteración: 121, Valor de la función: 0.9820198169648591
Iteración: 122, Valor de la función: 0.9820198169648591
Iteración: 123, Valor de la función: 0.9793557618311373
Iteración: 124, Valor de la función: 0.9793557618311373
Iteración: 125, Valor de la función: 0.9793557618311373
Iteración: 126, Valor de la función: 0.9793557618311373
Iteración: 127, Valor de la función: 0.9793557618311373
Iteración: 128, Valor de la función: 0.9793557618311373
Iteración: 129, Valor de la función: 0.9783790024895445
Iteración: 130, Valor de la función: 0.9783790024895445
Iteración: 131, Valor de la función: 0.9783790024895445
Iteración: 132, Valor de la función: 0.9783790024895445
Iteración: 133, Valor de la función: 0.9783790024895445
Iteración: 134, Valor de la función: 0.9783790024895445
Iteración: 135, Valor de la función: 0.9783790024895445
Iteración: 136, Valor de la función: 0.9783790024895445
Iteración: 137, Valor de la función: 0.9762082965604891
Iteración: 138, Valor de la función: 0.9762082965604891
Iteración: 139, Valor de la función: 0.9762082965604891
Iteración: 140, Valor de la función: 0.9762082965604891
```

```
Iteración: 141, Valor de la función: 0.9762082965604891
Iteración: 142, Valor de la función: 0.9762082965604891
Iteración: 143, Valor de la función: 0.9762082965604891
Iteración: 144, Valor de la función: 0.9762082965604891
Iteración: 145, Valor de la función: 0.9762082965604891
Iteración: 146, Valor de la función: 0.9762082965604891
Iteración: 147, Valor de la función: 0.9762082965604891
Iteración: 148, Valor de la función: 0.9762082965604891
Iteración: 149, Valor de la función: 0.9762082965604891
Iteración: 150, Valor de la función: 0.9762082965604891
Iteración: 151, Valor de la función: 0.9762082965604891
Iteración: 152, Valor de la función: 0.9762082965604891
Iteración: 153, Valor de la función: 0.9762082965604891
Iteración: 154, Valor de la función: 0.9762082965604891
Iteración: 155, Valor de la función: 0.9762082965604891
Iteración: 156, Valor de la función: 0.9762082965604891
Iteración: 157, Valor de la función: 0.9762082965604891
Iteración: 158, Valor de la función: 0.9762082965604891
Iteración: 159, Valor de la función: 0.9762082965604891
Iteración: 160, Valor de la función: 0.9762082965604891
Iteración: 161, Valor de la función: 0.9762082965604891
Iteración: 162, Valor de la función: 0.9762082965604891
Iteración: 163, Valor de la función: 0.9762082965604891
Iteración: 164, Valor de la función: 0.9762082965604891
Iteración: 165, Valor de la función: 0.9762082965604891
Iteración: 166, Valor de la función: 0.9762082965604891
Iteración: 167, Valor de la función: 0.9762082965604891
Iteración: 168, Valor de la función: 0.9762082965604891
Iteración: 169, Valor de la función: 0.9755136458700667
Iteración: 170, Valor de la función: 0.9755136458700667
Iteración: 171, Valor de la función: 0.9755136458700667
Iteración: 172, Valor de la función: 0.9755136458700667
Iteración: 173, Valor de la función: 0.9755136458700667
Iteración: 174, Valor de la función: 0.9755136458700667
Iteración: 175, Valor de la función: 0.9755136458700667
Iteración: 176, Valor de la función: 0.9755136458700667
Iteración: 177, Valor de la función: 0.9755136458700667
Iteración: 178, Valor de la función: 0.9755136458700667
Iteración: 179, Valor de la función: 0.9755136458700667
Iteración: 180, Valor de la función: 0.9755136458700667
Iteración: 181, Valor de la función: 0.9755136458700667
Iteración: 182, Valor de la función: 0.9755136458700667
Iteración: 183, Valor de la función: 0.9755136458700667
Iteración: 184, Valor de la función: 0.9755136458700667
Iteración: 185, Valor de la función: 0.9755136458700667
Iteración: 186, Valor de la función: 0.9755136458700667
Iteración: 187, Valor de la función: 0.9755136458700667
Iteración: 188, Valor de la función: 0.9755136458700667
```

```
Iteración: 189, Valor de la función: 0.9755136458700667
Iteración: 190, Valor de la función: 0.9736556775840517
Iteración: 191, Valor de la función: 0.9736556775840517
Iteración: 192, Valor de la función: 0.9736556775840517
Iteración: 193, Valor de la función: 0.9736556775840517
Iteración: 194, Valor de la función: 0.9736556775840517
Iteración: 195, Valor de la función: 0.9736556775840517
Iteración: 196, Valor de la función: 0.9736556775840517
Iteración: 197, Valor de la función: 0.9736556775840517
Iteración: 198, Valor de la función: 0.9736556775840517
Iteración: 199, Valor de la función: 0.9736556775840517
Iteración: 200, Valor de la función: 0.9736556775840517
Iteración: 201, Valor de la función: 0.9736556775840517
Iteración: 202, Valor de la función: 0.9736556775840517
Iteración: 203, Valor de la función: 0.9736556775840517
Iteración: 204, Valor de la función: 0.9736556775840517
Iteración: 205, Valor de la función: 0.9736556775840517
Iteración: 206, Valor de la función: 0.9736556775840517
Iteración: 207, Valor de la función: 0.9736556775840517
Iteración: 208, Valor de la función: 0.9736556775840517
Iteración: 209, Valor de la función: 0.9736556775840517
Iteración: 210, Valor de la función: 0.9736556775840517
Iteración: 211, Valor de la función: 0.9736556775840517
Iteración: 212, Valor de la función: 0.9736556775840517
Iteración: 213, Valor de la función: 0.9736556775840517
Iteración: 214, Valor de la función: 0.9736556775840517
Iteración: 215, Valor de la función: 0.9736556775840517
Iteración: 216, Valor de la función: 0.9736556775840517
Iteración: 217, Valor de la función: 0.9736556775840517
Iteración: 218, Valor de la función: 0.9736556775840517
Iteración: 219, Valor de la función: 0.9736556775840517
Iteración: 220, Valor de la función: 0.9736556775840517
Iteración: 221, Valor de la función: 0.9736556775840517
Iteración: 222, Valor de la función: 0.9736556775840517
Iteración: 223, Valor de la función: 0.9736556775840517
Iteración: 224, Valor de la función: 0.9736556775840517
Iteración: 225, Valor de la función: 0.9736556775840517
Iteración: 226, Valor de la función: 0.9736556775840517
Iteración: 227, Valor de la función: 0.9736556775840517
Iteración: 228, Valor de la función: 0.9736556775840517
Iteración: 229, Valor de la función: 0.9736556775840517
Iteración: 230, Valor de la función: 0.9736556775840517
Iteración: 231, Valor de la función: 0.9736556775840517
Iteración: 232, Valor de la función: 0.9736556775840517
Iteración: 233, Valor de la función: 0.9736556775840517
Iteración: 234, Valor de la función: 0.9736556775840517
Iteración: 235, Valor de la función: 0.9729415933370001
Iteración: 236, Valor de la función: 0.9729415933370001
```

```
Iteración: 237, Valor de la función: 0.9729415933370001
Iteración: 238, Valor de la función: 0.9729415933370001
Iteración: 239, Valor de la función: 0.9729415933370001
Iteración: 240, Valor de la función: 0.9729415933370001
Iteración: 241, Valor de la función: 0.9729415933370001
Iteración: 242, Valor de la función: 0.9729415933370001
Iteración: 243, Valor de la función: 0.9729415933370001
Iteración: 244, Valor de la función: 0.9729415933370001
Iteración: 245, Valor de la función: 0.9729415933370001
Iteración: 246, Valor de la función: 0.9729415933370001
Iteración: 247, Valor de la función: 0.9729415933370001
Iteración: 248, Valor de la función: 0.9729415933370001
Iteración: 249, Valor de la función: 0.9729415933370001
Iteración: 250, Valor de la función: 0.9729415933370001
Iteración: 251, Valor de la función: 0.9729415933370001
Iteración: 252, Valor de la función: 0.9729415933370001
Iteración: 253, Valor de la función: 0.9729415933370001
Iteración: 254, Valor de la función: 0.9729415933370001
Iteración: 255, Valor de la función: 0.9729415933370001
Iteración: 256, Valor de la función: 0.9729415933370001
Iteración: 257, Valor de la función: 0.9729415933370001
Iteración: 258, Valor de la función: 0.9728973657259959
Iteración: 259, Valor de la función: 0.9728973657259959
Iteración: 260, Valor de la función: 0.9728973657259959
Iteración: 261, Valor de la función: 0.9728973657259959
Iteración: 262, Valor de la función: 0.9728973657259959
Iteración: 263, Valor de la función: 0.9723701759857184
Iteración: 264, Valor de la función: 0.9723701759857184
Iteración: 265, Valor de la función: 0.9723701759857184
Iteración: 266, Valor de la función: 0.9718424950719413
Iteración: 267, Valor de la función: 0.9718424950719413
Iteración: 268, Valor de la función: 0.9718424950719413
Iteración: 269, Valor de la función: 0.9718424950719413
Iteración: 270, Valor de la función: 0.9718424950719413
Iteración: 271, Valor de la función: 0.9718424950719413
Iteración: 272, Valor de la función: 0.9718424950719413
Iteración: 273, Valor de la función: 0.9718424950719413
Iteración: 274, Valor de la función: 0.9718424950719413
Iteración: 275, Valor de la función: 0.9718424950719413
Iteración: 276, Valor de la función: 0.9718424950719413
Iteración: 277, Valor de la función: 0.9718424950719413
Iteración: 278, Valor de la función: 0.9718424950719413
Iteración: 279, Valor de la función: 0.9718424950719413
Iteración: 280, Valor de la función: 0.9718424950719413
Iteración: 281, Valor de la función: 0.9718424950719413
Iteración: 282, Valor de la función: 0.9718424950719413
Iteración: 283, Valor de la función: 0.9718424950719413
Iteración: 284, Valor de la función: 0.9718424950719413
```

```
Iteración: 285, Valor de la función: 0.9718424950719413
Iteración: 286, Valor de la función: 0.971836064442321
Iteración: 287, Valor de la función: 0.9715359763659907
Iteración: 288, Valor de la función: 0.9715359763659907
Iteración: 289, Valor de la función: 0.9715359763659907
Iteración: 290, Valor de la función: 0.9715359763659907
Iteración: 291, Valor de la función: 0.9715359763659907
Iteración: 292, Valor de la función: 0.9715359763659907
Iteración: 293, Valor de la función: 0.9715359763659907
Iteración: 294, Valor de la función: 0.9715359763659907
Iteración: 295, Valor de la función: 0.9715359763659907
Iteración: 296, Valor de la función: 0.9715359763659907
Iteración: 297, Valor de la función: 0.9715359763659907
Iteración: 298, Valor de la función: 0.9715359763659907
Iteración: 299, Valor de la función: 0.9715359763659907
Iteración: 300, Valor de la función: 0.9715359763659907
Iteración: 301, Valor de la función: 0.9715359763659907
Iteración: 302, Valor de la función: 0.9715359763659907
Iteración: 303, Valor de la función: 0.9715359763659907
Iteración: 304, Valor de la función: 0.9715359763659907
Iteración: 305, Valor de la función: 0.9715359763659907
Iteración: 306, Valor de la función: 0.9697393667702907
Iteración: 307, Valor de la función: 0.9697393667702907
Iteración: 308, Valor de la función: 0.9697393667702907
Iteración: 309, Valor de la función: 0.9697393667702907
Iteración: 310, Valor de la función: 0.9697393667702907
Iteración: 311, Valor de la función: 0.9697393667702907
Iteración: 312, Valor de la función: 0.9697393667702907
Iteración: 313, Valor de la función: 0.9697393667702907
Iteración: 314, Valor de la función: 0.9697393667702907
Iteración: 315, Valor de la función: 0.9697393667702907
Iteración: 316, Valor de la función: 0.9697393667702907
Iteración: 317, Valor de la función: 0.9697393667702907
Iteración: 318, Valor de la función: 0.9697393667702907
Iteración: 319, Valor de la función: 0.9697393667702907
Iteración: 320, Valor de la función: 0.9697393667702907
Iteración: 321, Valor de la función: 0.9697393667702907
Iteración: 322, Valor de la función: 0.9697393667702907
Iteración: 323, Valor de la función: 0.9697393667702907
Iteración: 324, Valor de la función: 0.9690029772585154
Iteración: 325, Valor de la función: 0.9690029772585154
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[930], line 11
      7 z = np.linspace(0, 1, n)
      9 bounds = [(F_0-0.5, F_0+0.5) for _ in range(n-2)]
```

```
---> 11 sol_sop = differential_evolution(
     12     area_func,
     13     bounds=bounds,
     14     maxiter=100000,
     15     popsize=4*n,
     16     mutation=(0.5, 1),
     17     recombination=0.7,
     18     strategy='best1bin',
     19     tol=0.01,
     20     callback=logger
     21 )

Cell In[871], line 20, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     18 def wrapped_func(*args, **kwargs):
     19     self.reset()
---> 20     return func(*args, **kwargs)

Cell In[871], line 20, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     18 def wrapped_func(*args, **kwargs):
     19     self.reset()
---> 20     return func(*args, **kwargs)

Cell In[871], line 20, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     18 def wrapped_func(*args, **kwargs):
     19     self.reset()
---> 20     return func(*args, **kwargs)

Cell In[408], line 52, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     50 def wrapped_func(*args, **kwargs):
     51     self.reset()
---> 52     return func(*args, **kwargs)

Cell In[406], line 22, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     20 def wrapped_func(*args, **kwargs):
     21     self.reset()
---> 22     return func(*args, **kwargs)

Cell In[871], line 20, in ObjectiveLogger.log_and_reset.<locals>.
↪wrapped_func(*args, **kwargs)
     18 def wrapped_func(*args, **kwargs):
     19     self.reset()
---> 20     return func(*args, **kwargs)
```

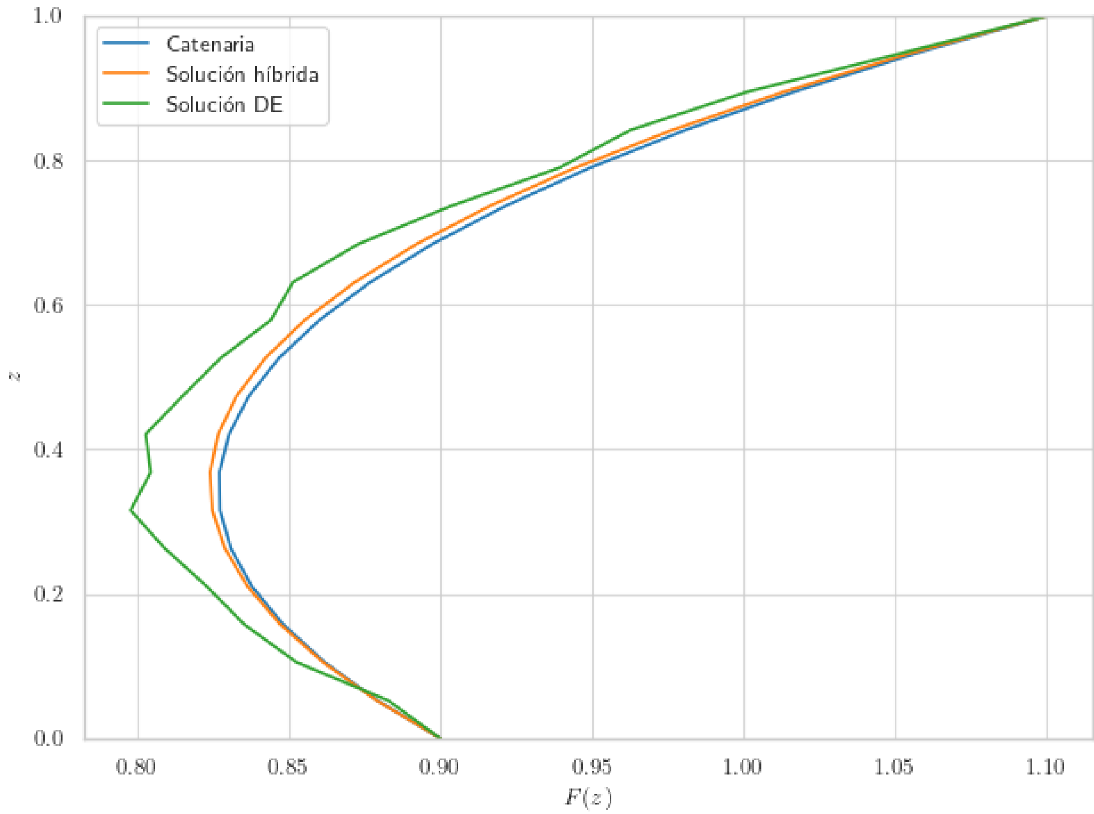
```
File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:502, in differential_evolution(func, bounds, args, strategy, maxiter,␣
↪popsize, tol, mutation, recombination, seed, callback, disp, polish, init,␣
↪atol, updating, workers, constraints, x0, integrality, vectorized)
    485 # using a context manager means that any created Pool objects are
    486 # cleared up.
    487 with DifferentialEvolutionSolver(func, bounds, args=args,
    488                                  strategy=strategy,
    489                                  maxiter=maxiter,
   (...)
    500                                  integrality=integrality,
    501                                  vectorized=vectorized) as solver:
--> 502     ret = solver.solve()
    504 return ret

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:1164, in DifferentialEvolutionSolver.solve(self)
   1161 for nit in range(1, self.maxiter + 1):
   1162     # evolve the population by a generation
   1163     try:
-> 1164         next(self)
   1165     except StopIteration:
   1166         warning_flag = True

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:1557, in DifferentialEvolutionSolver.__next__(self)
   1554     raise StopIteration
   1556 # create a trial solution
-> 1557 trial = self._mutate(candidate)
   1559 # ensuring that it's in the range [0, 1)
   1560 self._ensure_constraint(trial)

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:1688, in DifferentialEvolutionSolver._mutate(self, candidate)
   1685     bprime = self.mutation_func(candidate,
   1686                                 self._select_samples(candidate, 5))
   1687 else:
-> 1688     bprime = self.mutation_func(self._select_samples(candidate, 5))
   1690 if self.strategy in self._binomial:
   1691     crossovers = rng.uniform(size=self.parameter_count)

File c:
↪\Users\ismag\anaconda3\envs\ismael\Lib\site-packages\scipy\optimize\_different alevolution.
↪py:1765, in DifferentialEvolutionSolver._select_samples(self, candidate,␣
↪number_samples)
   1760 def _select_samples(self, candidate, number_samples):
```

```
   1761         """
   1762         obtain random integers from range(self.num_population_members),
   1763         without replacement. You can't have the original candidate either.
   1764         """
-> 1765         pool = np.arange(self.num_population_members)
   1766         self.random_number_generator.shuffle(pool)
   1768         idxs = []

KeyboardInterrupt:
```

[ ]:
```python
F_hib_sop = np.concatenate([[F0], sol_sop.x, [F1]])
F_de_sop = np.concatenate([[F0], sol_sop.population[0], [F1]])

a_sol, k_sol = fsolve(params_catenaria, initial_guess)
F_cat = catenaria(z)

error_hib_sop = np.linalg.norm(F_hib_sop - F_cat)/np.linalg.norm(F_cat)
error_de_sop = np.linalg.norm(F_de_sop - F_cat)/np.linalg.norm(F_cat)

print('El error para DE es:', error_de_sop)
print('El error para la solución híbrida es:', error_hib_sop)
```

```
El error para DE es: 0.01990219644766154
El error para la solución híbrida es: 0.003793093173347161
```

[ ]:
```python
plt.plot(F_cat, z, label='Catenaria')
plt.plot(F_hib_sop, z, label='Solución híbrida')
plt.plot(F_de_sop, z, label='Solución DE')
plt.legend()
plt.xlabel('$F(z)$')
plt.ylabel('$z$')
plt.ylim(0, 1)
plt.tight_layout()  # Ajustar el layout para que no se solapen los elementos
# plt.title('Curva optimizada con soportes de diferente tamaño, para $ \epsilon␣
↪= {}$'.format(eps))
plt.savefig('Figuras/sol_eps_de.pdf', format='pdf')
plt.show()
```



1.6 Problema con restricciones de volumen, con soportes de diferente tamaño. QUITAR ESTE APARTADO

A mayores de la condición de que los soportes sean desiguales, se pasa a estudiar el efecto combinado de esta junto con la restricción de volmen.

[ ]:
```
F_0 = 1
F0 = F_0
F1 = F_0
n = 50
F_init = np.empty(n-2)
F_init.fill(F_0)
lb = np.zeros(n-2)
ub = np.ones(n-2) * np.inf
bounds = np.vstack((lb, ub)).T

z = np.linspace(0, 1, n)

sol_grad = minimize(area_func, F_init, bounds=bounds, method='SLSQP')
Famp_grad = np.concatenate(([F0], sol_grad.x, [F1]))
```

[ ]:
```
eps = 0.05
F0 = F_0 - eps
F1 = F_0 + eps

sol_grad_voleps = minimize(area_func, F_init, method='SLSQP', bounds=bounds)
Famp_voleps = np.concatenate(([F0], sol_grad_voleps.x, [F1]))
```

A fin de que los valores no difieran en exceso del valor obtenido del problema sin restricciones, primero se calcula el volumen de la solución con $F_0 = 1$, con la que se partirá para calcular la solución con restricciones.

[ ]:
```
V_lig = 0.0 # la ligadura de volumen

vol_eps = vol_func(sol_grad_voleps.x)
print('Volumen sin restricciones =', vol_eps)
```

```
Volumen sin restricciones = 0.8047913386876858
```

Así, se impone la mencionada restricción para observar su influencia en la solución del problema.

[ ]:
```
cons = ({'type': 'eq', 'fun': vol_func})
```

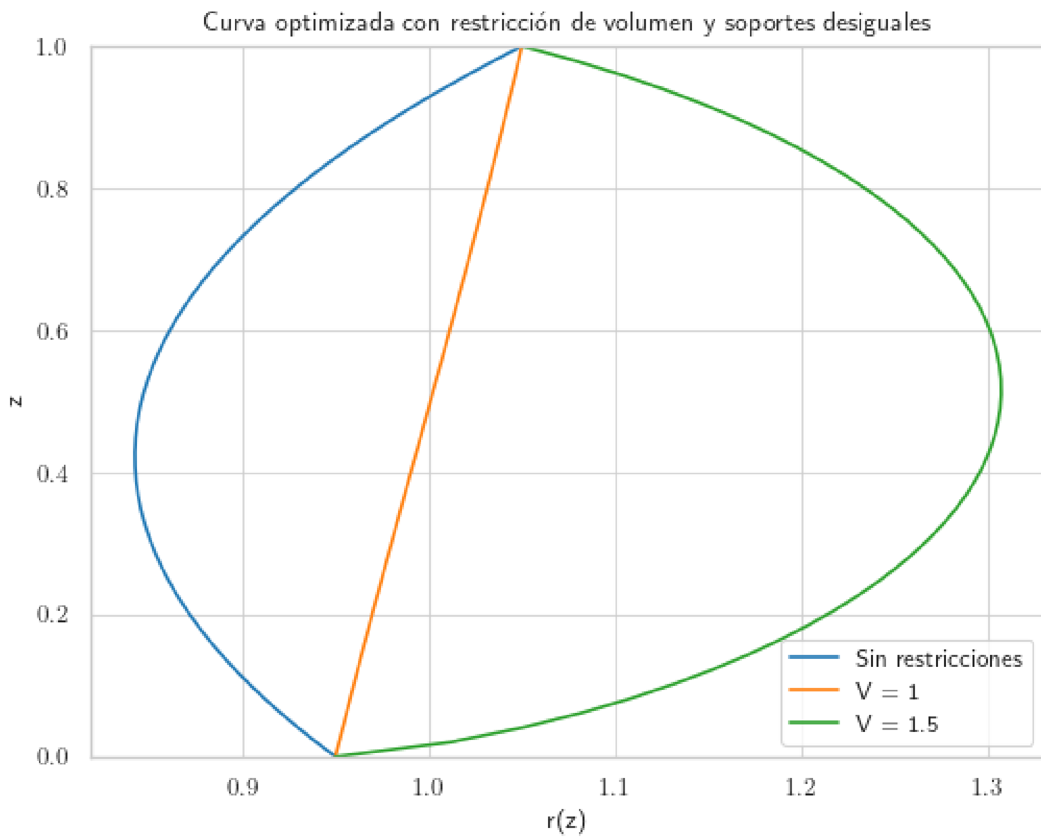
[ ]:
```
V_lig = 0.5 # inestable

sol_grad_vol05 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_vol05 = np.concatenate(([F0], sol_grad_vol05.x, [F1]))
```

[ ]:
```
V_lig = 1

sol_grad_vol1 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_vol1 = np.concatenate(([F0], sol_grad_vol1.x, [F1]))
```

[ ]:
```
V_lig = 1.5

sol_grad_vol15 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
 ↪bounds=bounds)
Famp_vol15 = np.concatenate(([F0], sol_grad_vol15.x, [F1]))
```

[ ]:
```
plt.plot(Famp_voleps, z, label='Sin restricciones')
plt.plot(Famp_vol1, z, label='V = 1')
plt.plot(Famp_vol15, z, label='V = 1.5')
plt.legend()
plt.xlabel('r(z)')
plt.ylabel('z')
plt.ylim(0, 1)
plt.title('Curva optimizada con restricción de volumen y soportes desiguales')
plt.grid(True)
```

```
plt.show()
```



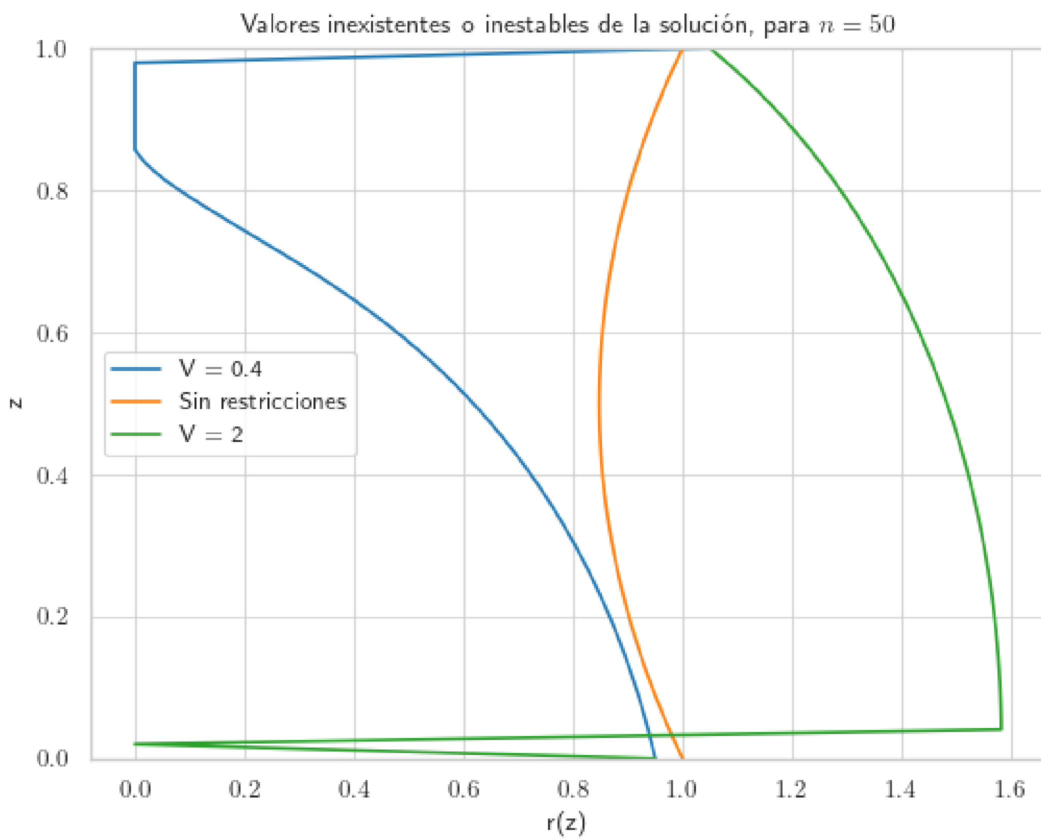
Como cabía de esperar, la forma de la solución se modifica al aumentar el valor de la restricción de volumen. Es importante recalcar que esta restricción no puede tomar cualquier valor, ya que si difiere en gran medida del volumen obtenido sin restricciones, la solución puede resultar inexistente o inestable.

Precisamente, sobre esas posibilidades se va a discutir a continuación.

[ ]:
```
V_lig = 2

sol_grad_vol2 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
↪bounds=bounds)
Famp_vol2 = np.concatenate(([F0], sol_grad_vol2.x, [F1]))
```

[ ]:
```
V_lig = 0.4

sol_grad_vol04 = minimize(area_func, F_init, constraints=cons, method='SLSQP',␣
↪bounds=bounds)
Famp_vol04 = np.concatenate(([F0], sol_grad_vol04.x, [F1]))
```

[ ]:
```
plt.plot(Famp_vol04, z, label='V = 0.4')
plt.plot(Famp_grad, z, label='Sin restricciones')
plt.plot(Famp_vol2, z, label='V = 2')
plt.legend()
plt.xlabel('r(z)')
plt.ylabel('z')
plt.ylim(0, 1)
plt.title('Valores inexistentes o inestables de la solución, para $n = {}$'.
 ↪format(n))
plt.grid(True)
plt.show()
```



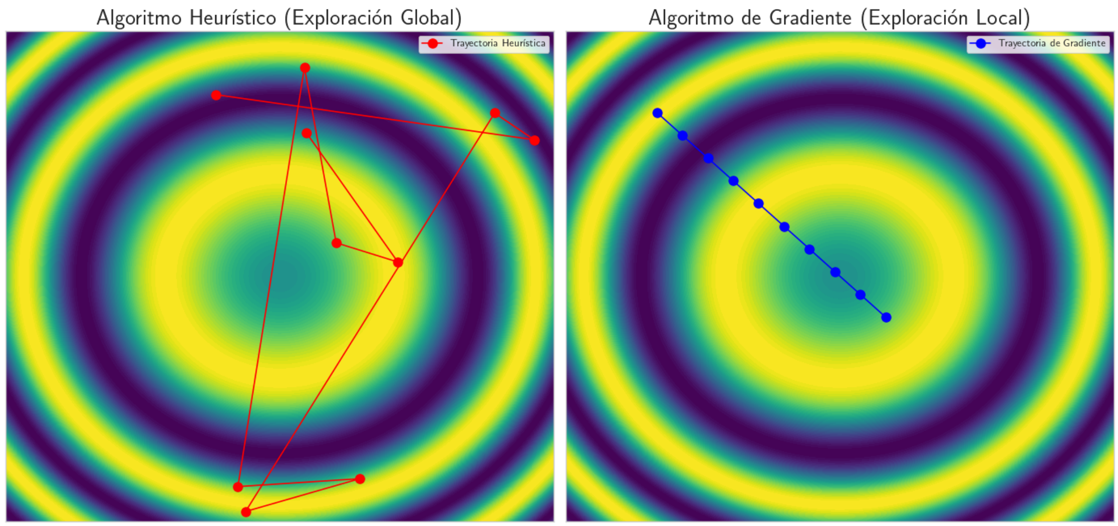
Se aprecia que, para valores que se alejan en gran medida de la solución sin restricciones, se generan soluciones que, como poco, no tienen sentido físico.

Cabe pensar que, para los casos más sensibles, pueda existir solución solo que el método no tiene la precisión adecuada para calcularlo. Así, se plantea evaluar el efecto de aumentar el número de puntos, a fin de tener una representación más precisa, caso que se expone a continuación.

2 Código para generar archivos para la presentación

[68]:
```
# Configuración del espacio de búsqueda
x = np.linspace(-3, 3, 500)
y = np.linspace(-3, 3, 500)
X, Y = np.meshgrid(x, y)
Z = np.sin(X**2 + Y**2)

# Trajectorias de algoritmos
np.random.seed(0)
# Simulación de trayectoria heurística
heuristic_path_x = np.random.uniform(-3, 3, 10)
heuristic_path_y = np.random.uniform(-3, 3, 10)
heuristic_path_z = np.sin(heuristic_path_x**2 + heuristic_path_y**2)

# Simulación de trayectoria de gradiente
gradient_path_x = np.linspace(-2, 0.5, 10)
gradient_path_y = -gradient_path_x
gradient_path_z = np.sin(gradient_path_x**2 + gradient_path_y**2)

# Crear la figura
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 7))
# fig.suptitle('Comparación: Algoritmos Heurísticos vs. Algoritmos de␣
↪Gradiente', fontsize=16)

# Algoritmo Heurístico
ax1.contourf(X, Y, Z, levels=50, cmap='viridis')
ax1.plot(heuristic_path_x, heuristic_path_y, 'ro-', markersize=8,␣
↪label='Trayectoria Heurística')
ax1.set_title('Algoritmo Heurístico (Exploración Global)', fontsize=20)
ax1.set_xticks([])
ax1.set_yticks([])
# ax1.set_xlabel('$x$')
# ax1.set_ylabel('$y$')
ax1.legend()

# Algoritmo de Gradiente
ax2.contourf(X, Y, Z, levels=50, cmap='viridis')
ax2.plot(gradient_path_x, gradient_path_y, 'bo-', markersize=8,␣
↪label='Trayectoria de Gradiente')
ax2.set_title('Algoritmo de Gradiente (Exploración Local)', fontsize=20)
ax2.set_xticks([])
ax2.set_yticks([])
# ax2.set_xlabel('$x$')
# ax2.set_ylabel('$y$')
ax2.legend()
```

```
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.savefig('Figuras/Presentación/comparacion_heuristicos_gradiente.pdf',␣
 ↪format='pdf')
plt.show()
```



3 Aspectos a considerar para implementar al problema.

- Considerar las restricciones de volumen como el volumen de la solución analítica (ver si converge en más iteraciones o qué).