Modelización de una función de alto coste computacional con técnicas de inteligencia artificial.

Realizado por Jaime Gabriel Vegas.

Máster Universitario de Modelización Matemática.
Técnicas informáticas avanzadas para la modelización.

Curso 2023-2024.

VNiVERSiDAD D SALAMANCA

Índice.

Índice

Ecuación empleada.

Ecuación del "módulo de distancia":

$$\mu(z, H_0, \Omega_m, \Omega_\Lambda, \Omega_r) = -5 + 5 \log_{10} d_L((pc)) \tag{1}$$

donde

$$d_L(z, H_0, \Omega_m, \Omega_\Lambda, \Omega_r) = \frac{c(1+z)}{\sqrt{|\Omega_k^{(0)}|} H_0} S_k\left(\sqrt{|\Omega_k^{(0)}|} \int_0^z \frac{dz}{E(z)}\right) \tag{2}$$

Ecuación empleada

Donde, a su vez,

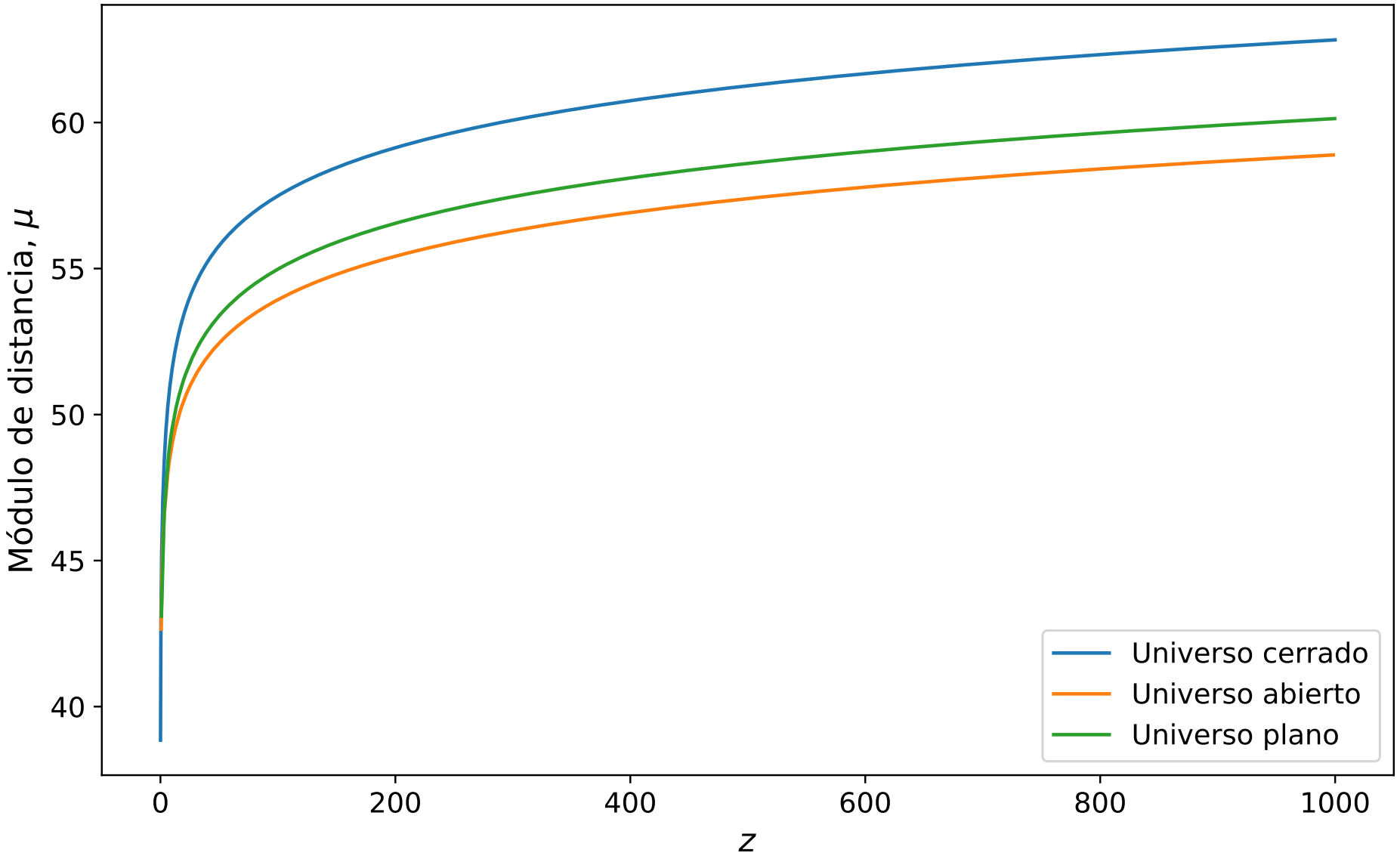
$$d_L(z, H_0, \Omega_m, \Omega_\Lambda, \Omega_r) = \frac{c(1+z)}{\sqrt{|\Omega_k^{(0)}|}H_0} S_k\left(\sqrt{|\Omega_k^{(0)}|}\int_0^z \frac{dz}{E(z)}\right), \tag{3}$$

$$S_k(x) = \begin{cases} \sin(x), & \Omega_k^{(0)} < 0 \\ x, & \Omega_k^{(0)} = 0 \\ \sinh(x), & \Omega_k^{(0)} > 0, \end{cases} \tag{4}$$

$$E(z) \equiv \frac{H(z)}{H_0} = \sqrt{\Omega_r^{(0)}(1+z)^4 + \Omega_m^{(0)}(1+z)^3 + \Omega_k^{(0)}(1+z)^2 + \Omega_\Lambda^{(0)}}, \tag{5}$$

$$\Omega_r^{(0)} + \Omega_m^{(0)} + \Omega_k^{(0)} + \Omega_\Lambda^{(0)} = 1. \tag{6}$$

Aspecto para valores concretos de Ω_i y H_0.



Primeros pasos.

- Vemos que para todos los casos la gráfica tiene la misma "forma".
- Por eso, hacemos el análisis para un caso concreto, ya que para el resto sería igual.

Nuestro "objetivo" será:

- Obtener 100000 puntos de la anterior ecuación.
- En hacer esto, Mathematica tarda alrededor de 5 minutos

Índice

Modelos empleados.

Emplearemos los siguientes modelos:

- SVR (Regresión de vectores de soporte).
  Encuentra el hiperplano que mejor ajusta las datos a partir de una determinada función.
- SGD Regressor (Regresión por descenso estocástico del gradiente).
  Ajusta unos pesos en función de la dirección del descenso del gradiente.
- KNeighborsRegressor (Vecinos más cercanos).
  Promedio de los k vecinos más cercanos.
- Multi-layer Perceptron Regressor. (Red Neuronal)
- Modelo en TensorFlow. (Red Neuronal)

Métricas.

- Error absoluto medio,

$$\text{MAE} = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n}. \tag{7}$$

- Error cuadrático medio,

$$\text{MSA} = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}. \tag{8}$$

- Coeficiente de determinación,

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{9}$$

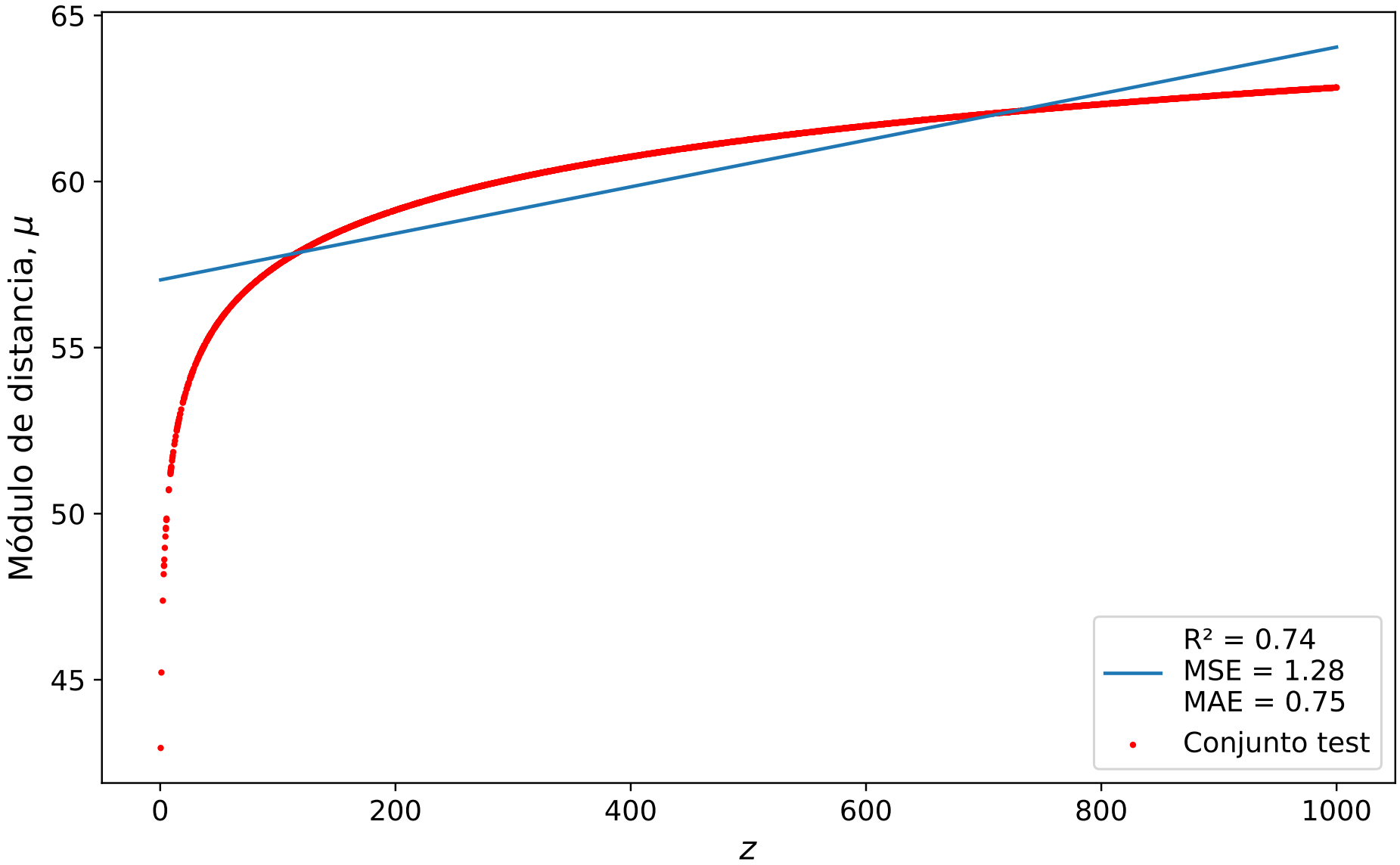
Índice

Entrenamiento y resultados.

- Empezamos con una muestra de 10000 valores de entrenamiento + test (alrededor de 35 segundos para Mathematica).
- Para este caso, fijaremos los valores de Ω_i y H_0, y solo variaremos z.
- Consideramos un modelo cerrado con

$$H_0 = 40\frac{\text{km}\cdot\text{s}}{\text{Mpc}}, \quad \Omega_m = \Omega_\Lambda = 0{,}3, \quad \Omega_r = 0. \tag{10}$$

Ejemplo de ajuste lineal.



División de los datos.

```
X_train, X_test, y_train, y_test = train_test_split(
    df_caso_particular_cerrado['vectorZ'].values,
    df_caso_particular_cerrado['resultados'].values,
    test_size=0.33,
    random_state=2024)
```

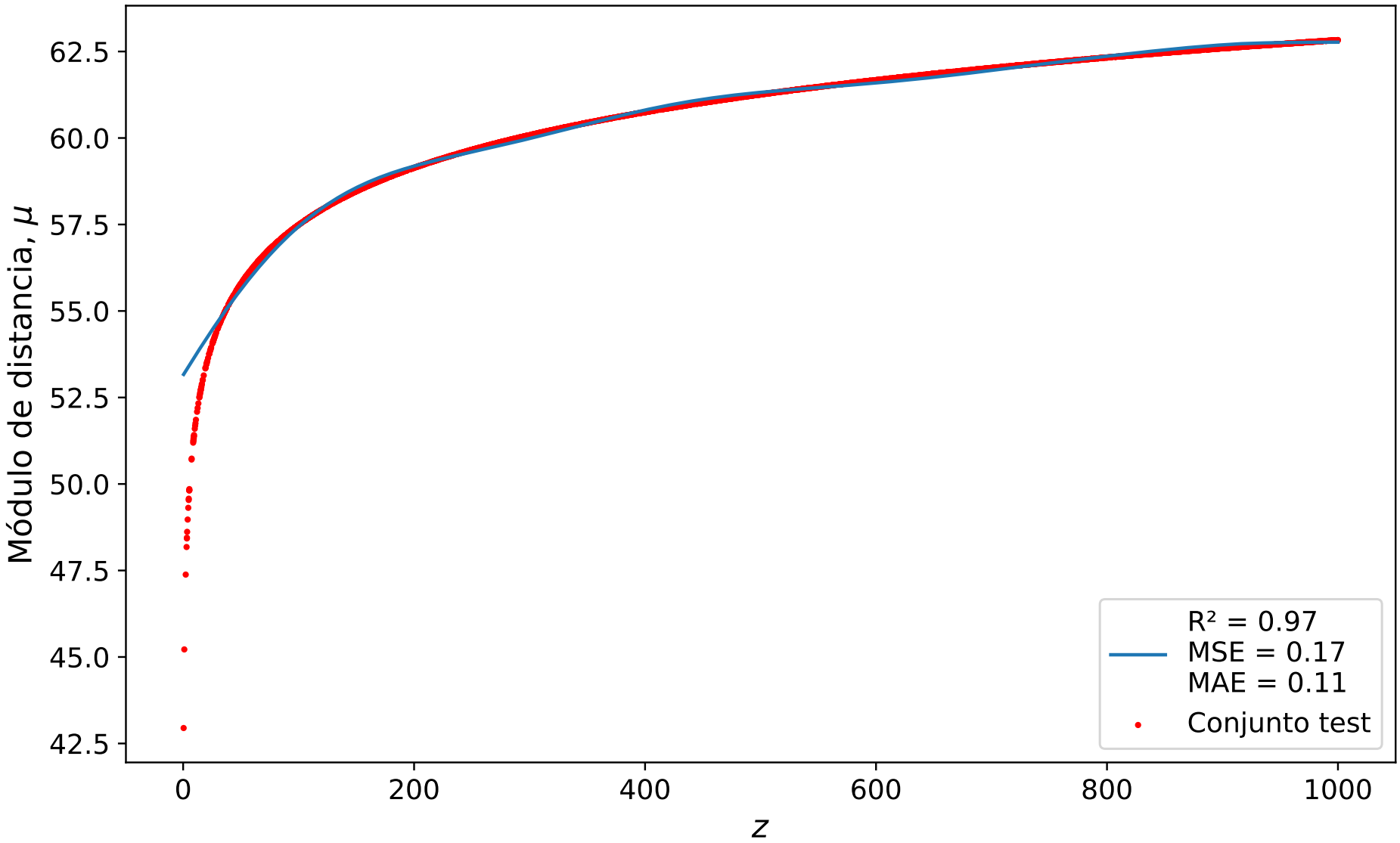
[164] ✓ 0.0s Python

Modelo SVR.

```python
# SVR

SVR_particular_cerrado = make_pipeline(StandardScaler(),
                                       SVR(verbose = True))


SVR_particular_cerrado.fit(X_train_particular_cerrado.reshape(-1, 1),
                           y_train_particular_cerrado)

prediccion_SVR_particular_cerrado = SVR_particular_cerrado.predict(X_test_particular_cerrado.reshape(-1, 1))

mse_SVR_particular_cerrado = mean_squared_error(y_test_particular_cerrado.reshape(-1, 1),
                                                prediccion_SVR_particular_cerrado)

r2_SVR_particular_cerrado = r2_score(y_test_particular_cerrado,
                                     prediccion_SVR_particular_cerrado)

print("\nMSE del modelo SVR:", mse_SVR_particular_cerrado)
print("R² del modelo SVR:", r2_SVR_particular_cerrado)
```

✓ 1.2s Python

Modelo SVR.

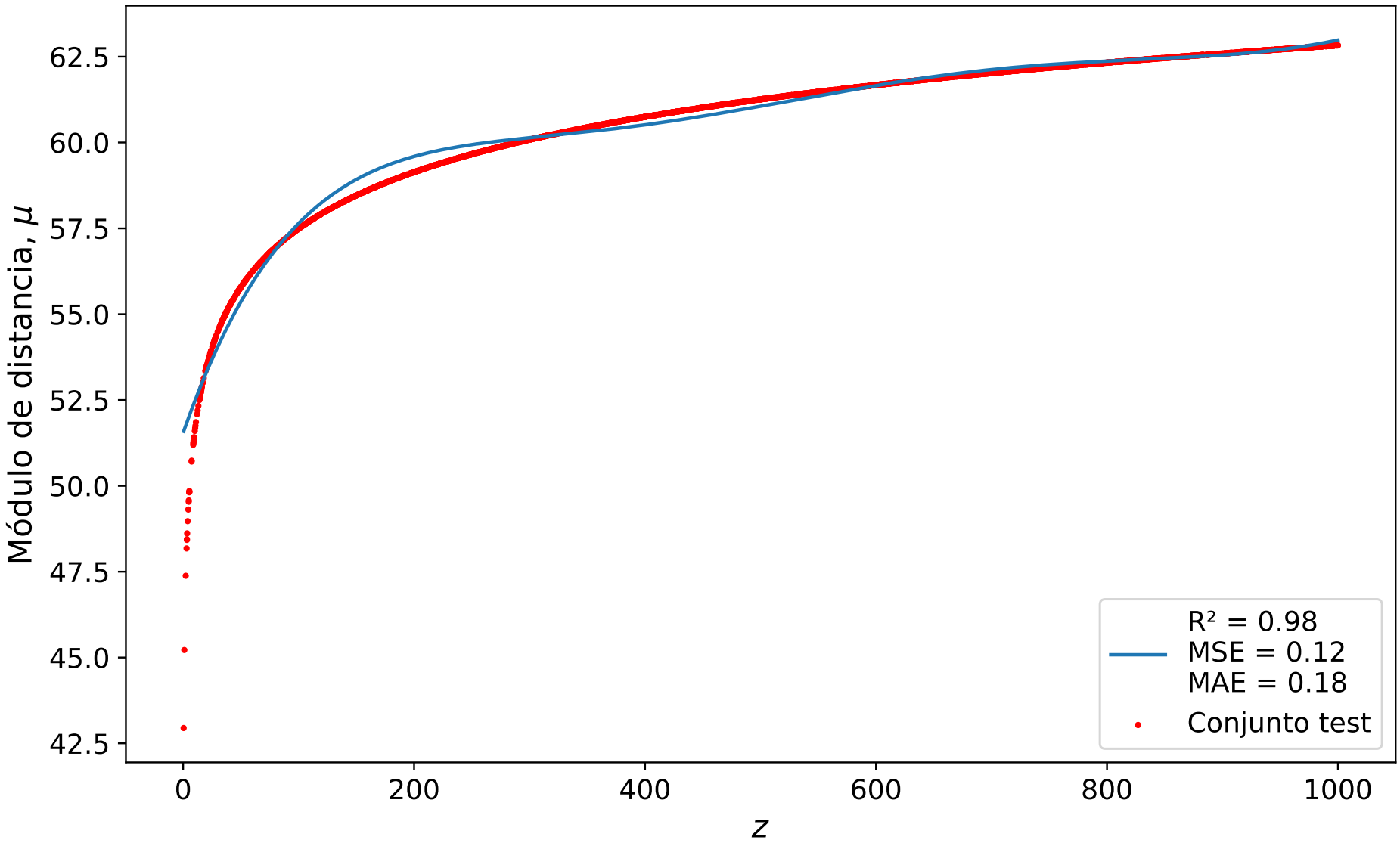

62.5
60.0
57.5
55.0
52.5
50.0
47.5
45.0
42.5
Módulo de distancia, μ
0
200
400
600
800
1000
z
R^2 = 0.97
MSE = 0.17
MAE = 0.11
Conjunto test

Modelo SGD.

```
# SGD
SGD_particular_cerrado = make_pipeline(StandardScaler(),
                                       PolynomialFeatures(degree=6),
                                       SGDRegressor(verbose = True))

SGD_particular_cerrado.fit(X_train_particular_cerrado.reshape(-1, 1),
                           y_train_particular_cerrado)

prediccion_SGD_particular_cerrado = SGD_particular_cerrado.predict(X_test_particular_cerrado.reshape(-1, 1))

score_SGD_particular_cerrado = SGD_particular_cerrado.score(X_test_particular_cerrado.reshape(-1, 1),
                                                            y_test_particular_cerrado)

mse_SVR_particular_cerrado = mean_squared_error(y_test_particular_cerrado.reshape(-1, 1),
                                                prediccion_SGD_particular_cerrado)

r2_SGD_particular_cerrado = r2_score(y_test_particular_cerrado,
                                     prediccion_SGD_particular_cerrado)
```

✓ 0.0s Python

Modelo SGD.

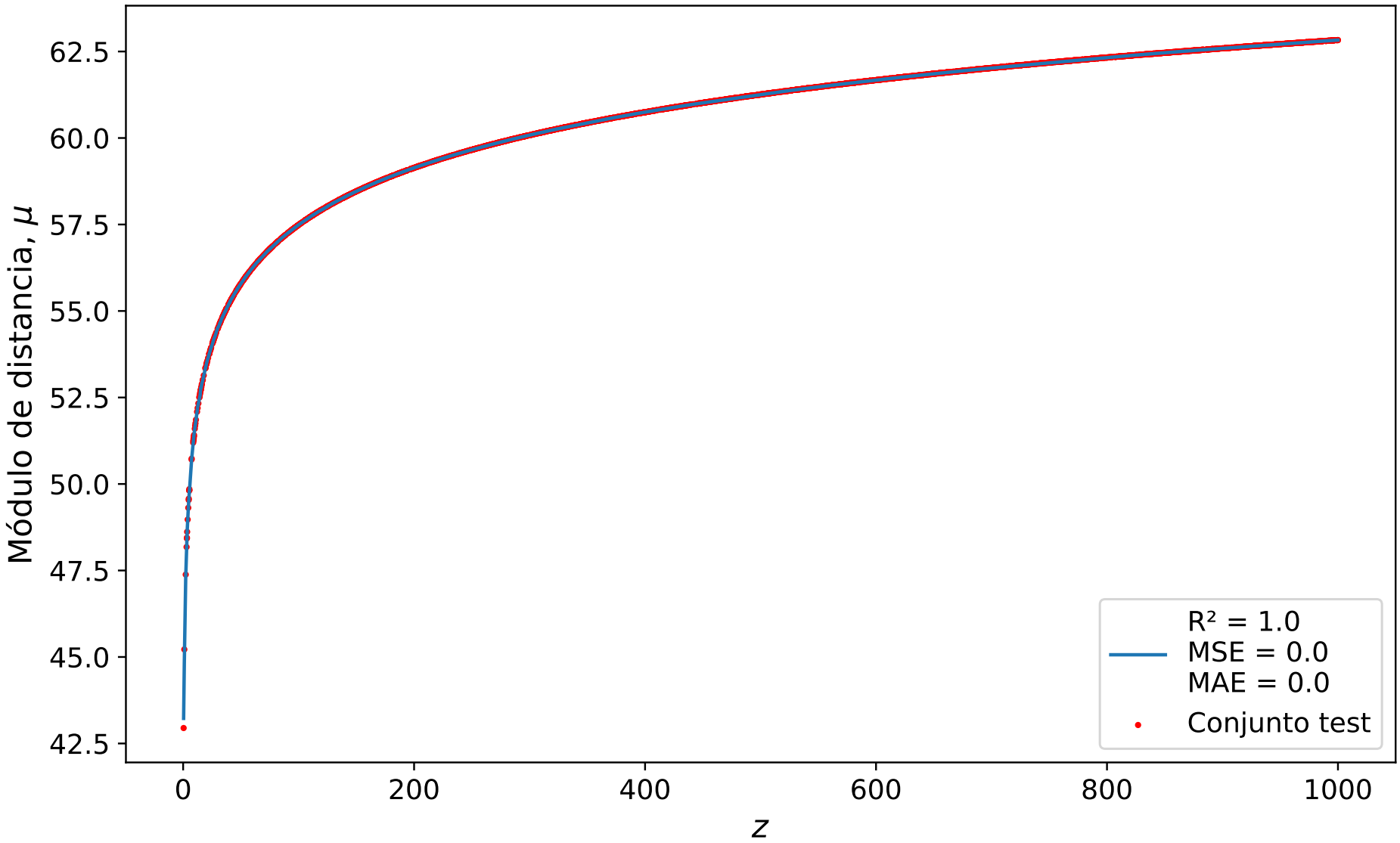

62.5
60.0
57.5
55.0
52.5
50.0
47.5
45.0
42.5
Módulo de distancia, μ
0
200
400
600
800
1000
z
R^2 = 0.98
MSE = 0.12
MAE = 0.18
Conjunto test

Modelo KNeighbors.

```python
# KNeighborsRegressor

KNeighbors_particular_cerrado = KNeighborsRegressor(n_neighbors=2)

KNeighbors_particular_cerrado.fit(X_train_particular_cerrado.reshape(-1, 1),
                                  y_train_particular_cerrado)

prediccion_KNeighbors_particular_cerrado = KNeighbors_particular_cerrado.predict(X_test_particular_cerrado.reshape(-1, 1))

mse_KNeighbors_particular_cerrado = mean_squared_error(y_test_particular_cerrado.reshape(-1, 1),
                                                       prediccion_KNeighbors_particular_cerrado)

r2_KNeighbors_particular_cerrado = r2_score(y_test_particular_cerrado, prediccion_KNeighbors_particular_cerrado)
```

✓ 0.0s Python

Modelo KNeighbors.

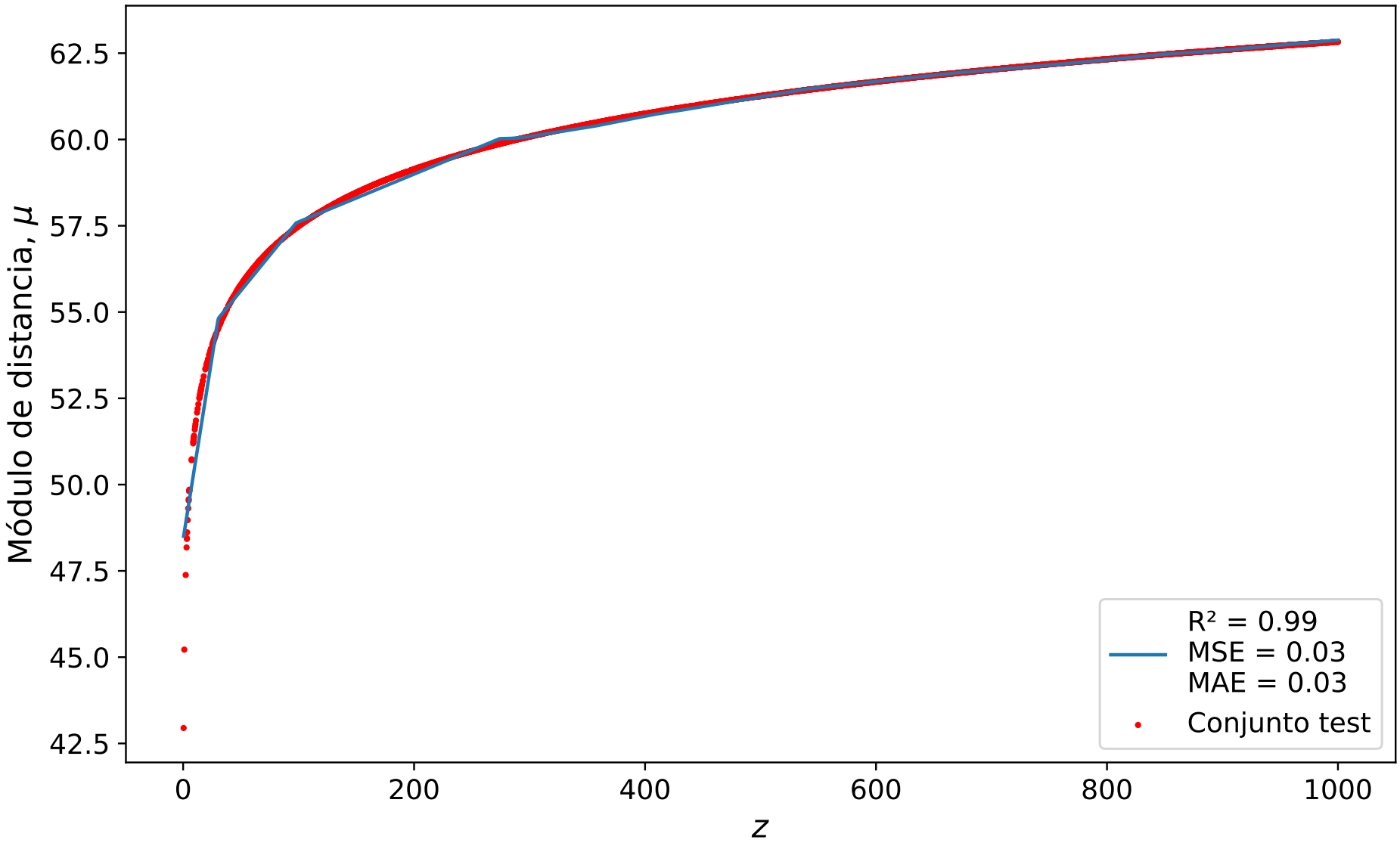


Modelo MLP.

```
# MLP

MLP_particular_cerrado = make_pipeline(StandardScaler(), MLPRegressor(hidden_layer_sizes=[100, 50],
                                                                      activation='relu',
                                                                      random_state = 2024))

MLP_particular_cerrado.fit(X_train_particular_cerrado.reshape(-1, 1),
                           y_train_particular_cerrado)

prediccion_MLP_particular_cerrado = MLP_particular_cerrado.predict(X_test_particular_cerrado.reshape(-1, 1))

mse_MLP_particular_cerrado = mean_squared_error(y_test_particular_cerrado.reshape(-1, 1),
                                                prediccion_MLP_particular_cerrado)

r2_MLP_particular_cerrado = r2_score(y_test_particular_cerrado,
                                     prediccion_MLP_particular_cerrado)
```

✓ 4.4s Python

Modelo MLP.

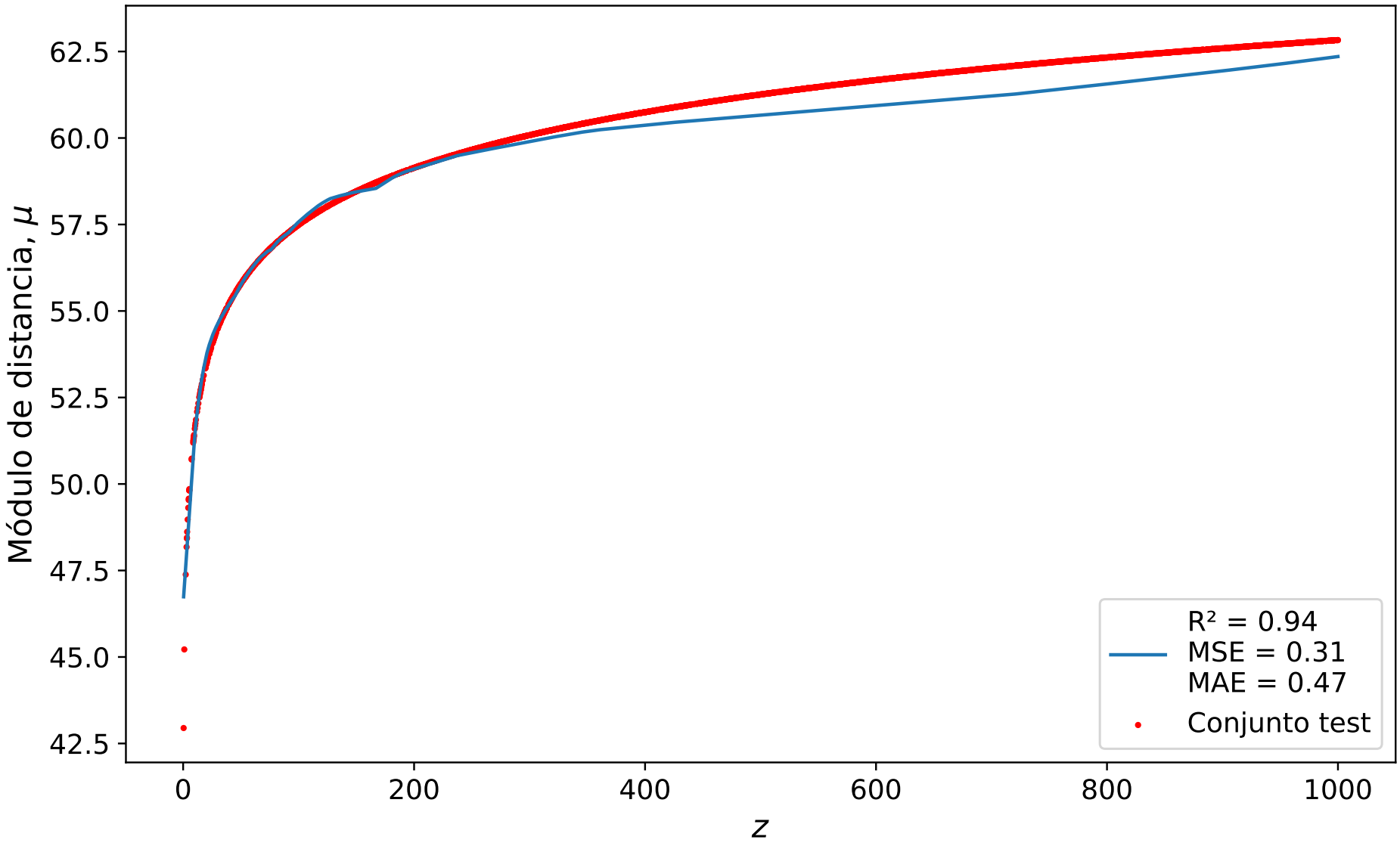

62.5
60.0
57.5
55.0
52.5
50.0
47.5
45.0
42.5
Módulo de distancia, μ
0
200
400
600
800
1000
z
R^2 = 0.99
MSE = 0.03
MAE = 0.03
Conjunto test

Modelo TensorFlow.

```python
# TensorFlow

tf_particular_cerrado = tf.keras.Sequential([
    tf.keras.layers.Dense(1000, activation='relu', input_shape=(1,)),
    #tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(500, activation='relu'),
    #tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(50, activation='relu'),
    #tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(1)  # Capa de salida para la regresión lineal
])

tf_particular_cerrado.compile(optimizer='adam',
                              loss='mean_squared_error',
                              metrics=['mae'])

tf_particular_cerrado.fit(X_train_particular_cerrado,
                          y_train_particular_cerrado,
                          validation_data = (X_test_particular_cerrado, y_test_particular_cerrado),
                          epochs = 10)
```

] ✓ 52.1s Python

Modelo TensorFlow.



Resumen

Modelo	Entrenamiento (s)	R^2	MSE	MAE
SVR	1 - 2	0,97	0,17	0,11
SGR	0	0,97	0,14	0,18
KNeighbors	0	1,0	0,0	0,0
MLP	4 - 7	0,99	0,06	0,03
TensorFlow	50 - 60	0,94	0,31	0,47

- El R^2, MSE y MSE están aproximados a dos cifras decimales.

Índice

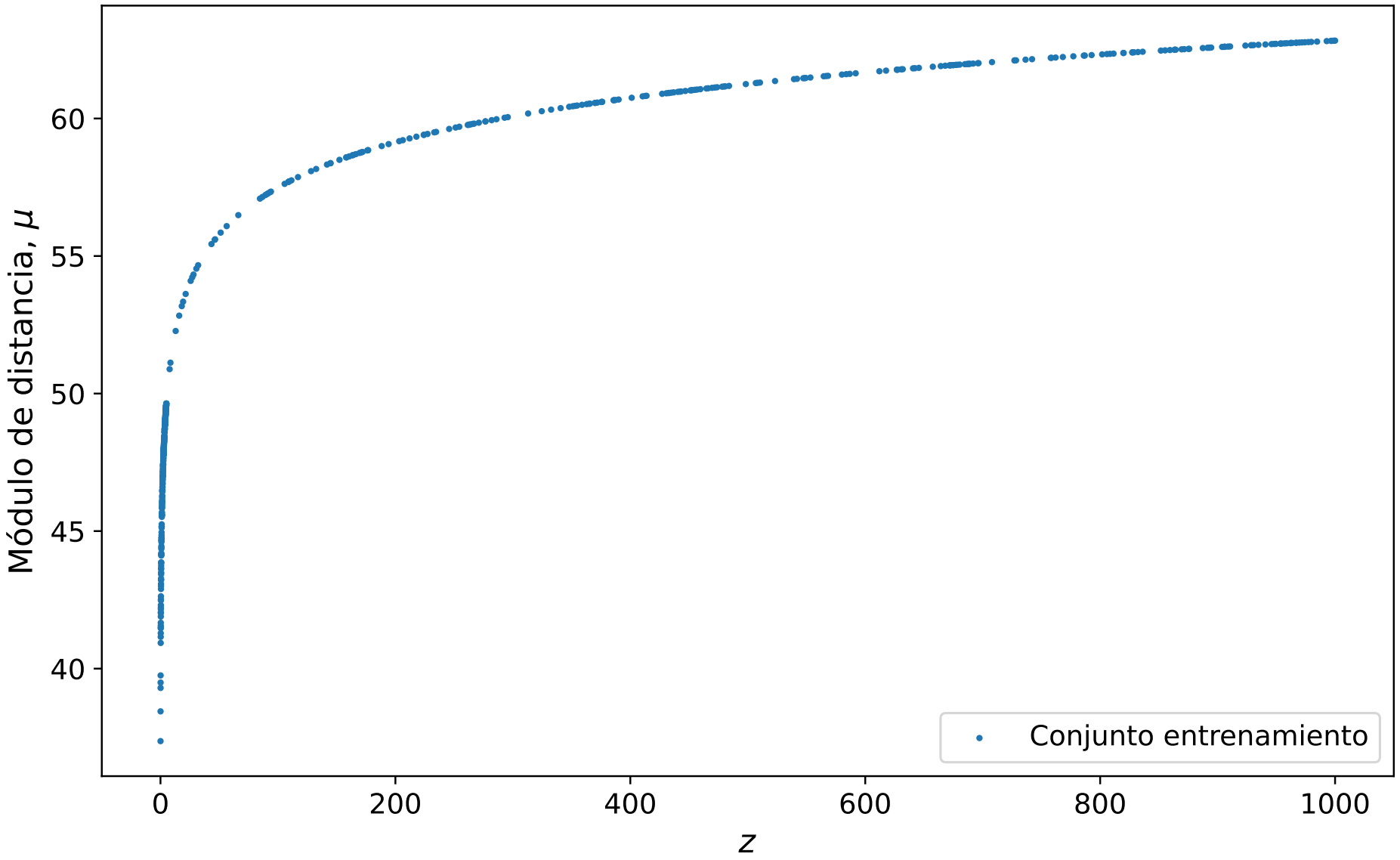
Cómo generar 100000 puntos lo más rápido posible.

Para generar resultados para valores arbitrarios de H_0 y Ω_i lo que hacemos es:

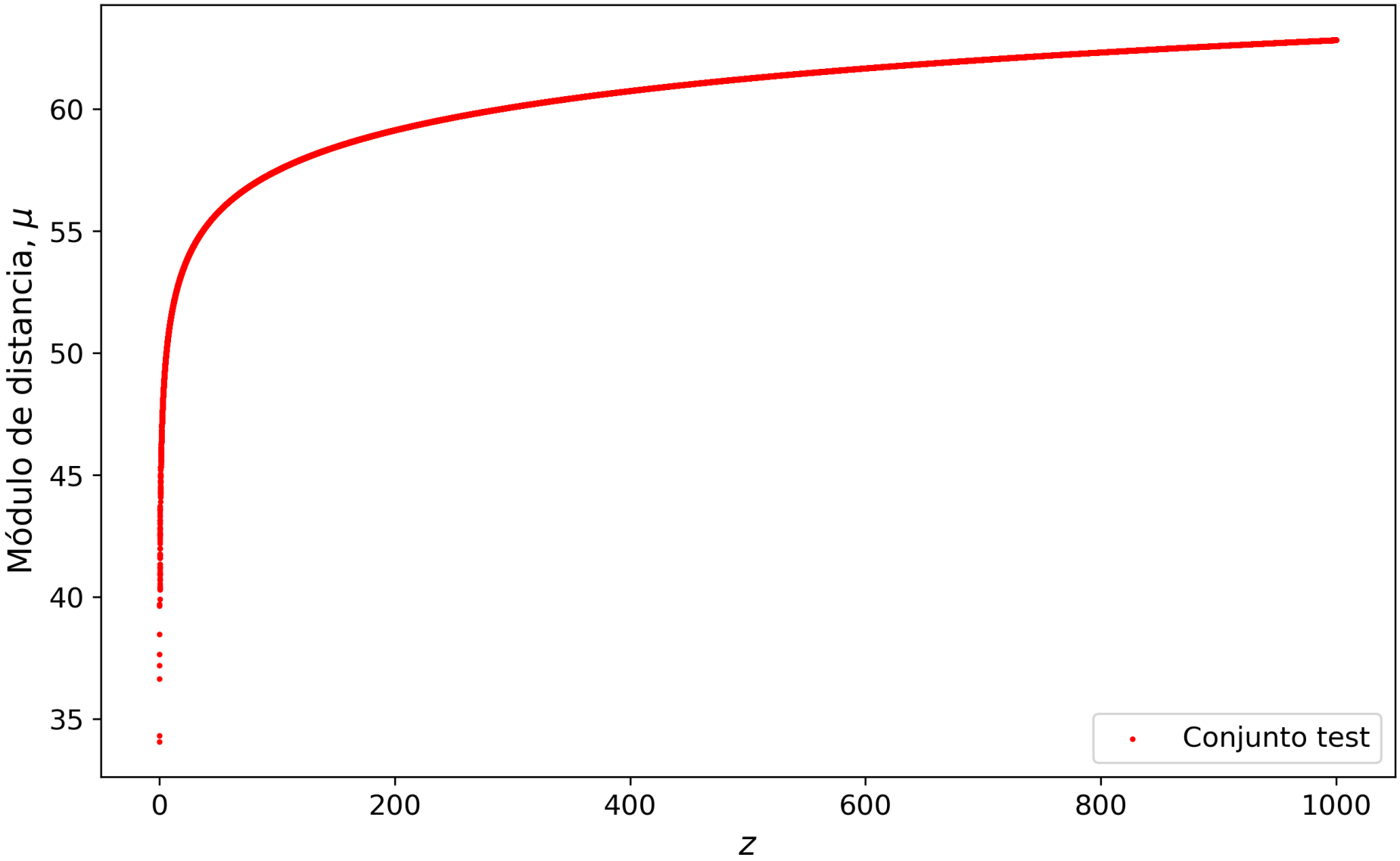
- Generar, para los valores de H_0 y Ω_i que queremos, unos pocos puntos (500) correctamente distribuidos.
- Entrenar al modelo KNeighbors en ese conjunto de datos.
- Generar el resto de datos a partir del modelo.

Tiempo de cálculo de los 100000 valores en Mathematica: 4 minutos 40 segundos.

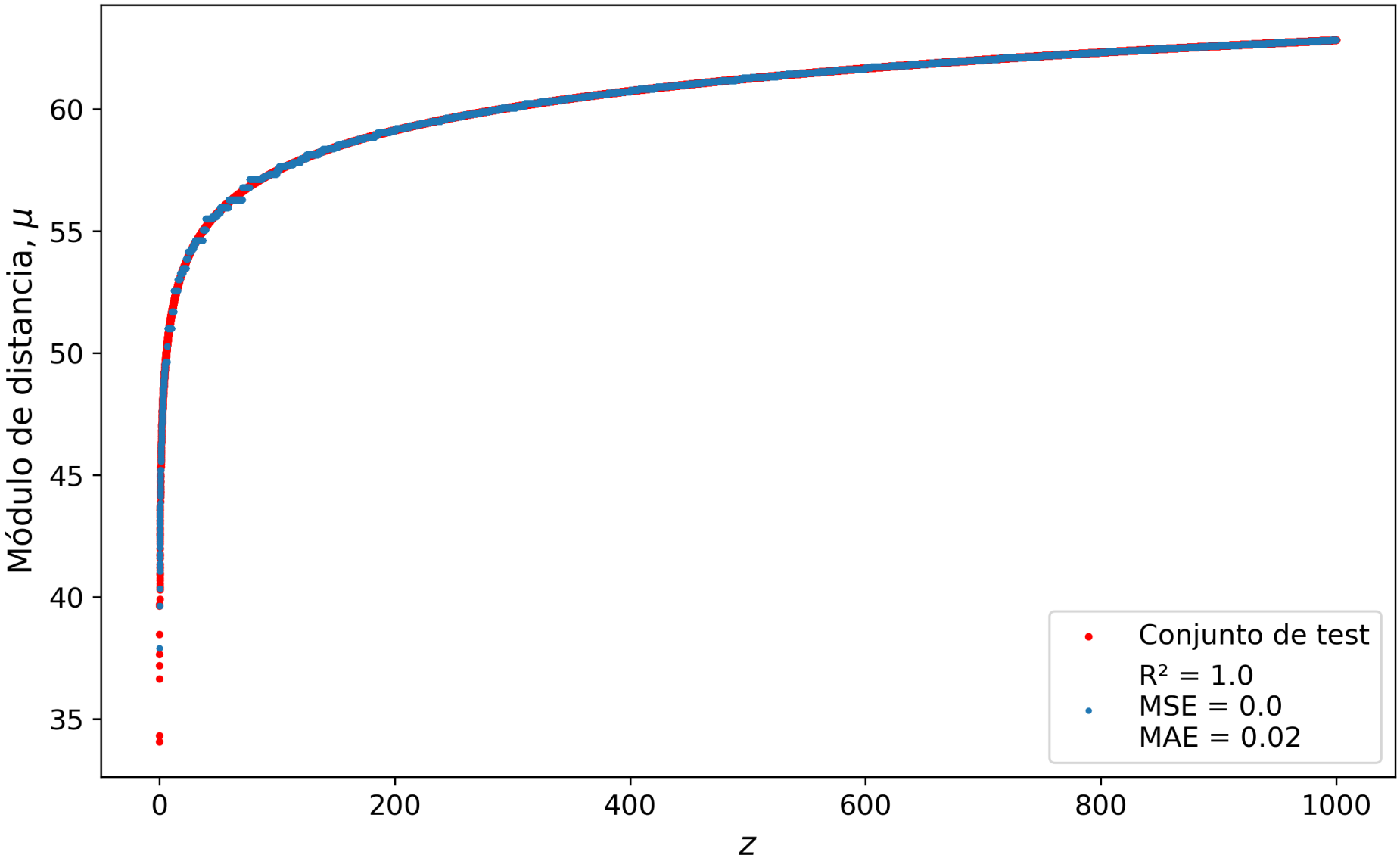
Conjunto de entrenamiento.



Conjunto de test.



Datos generados.



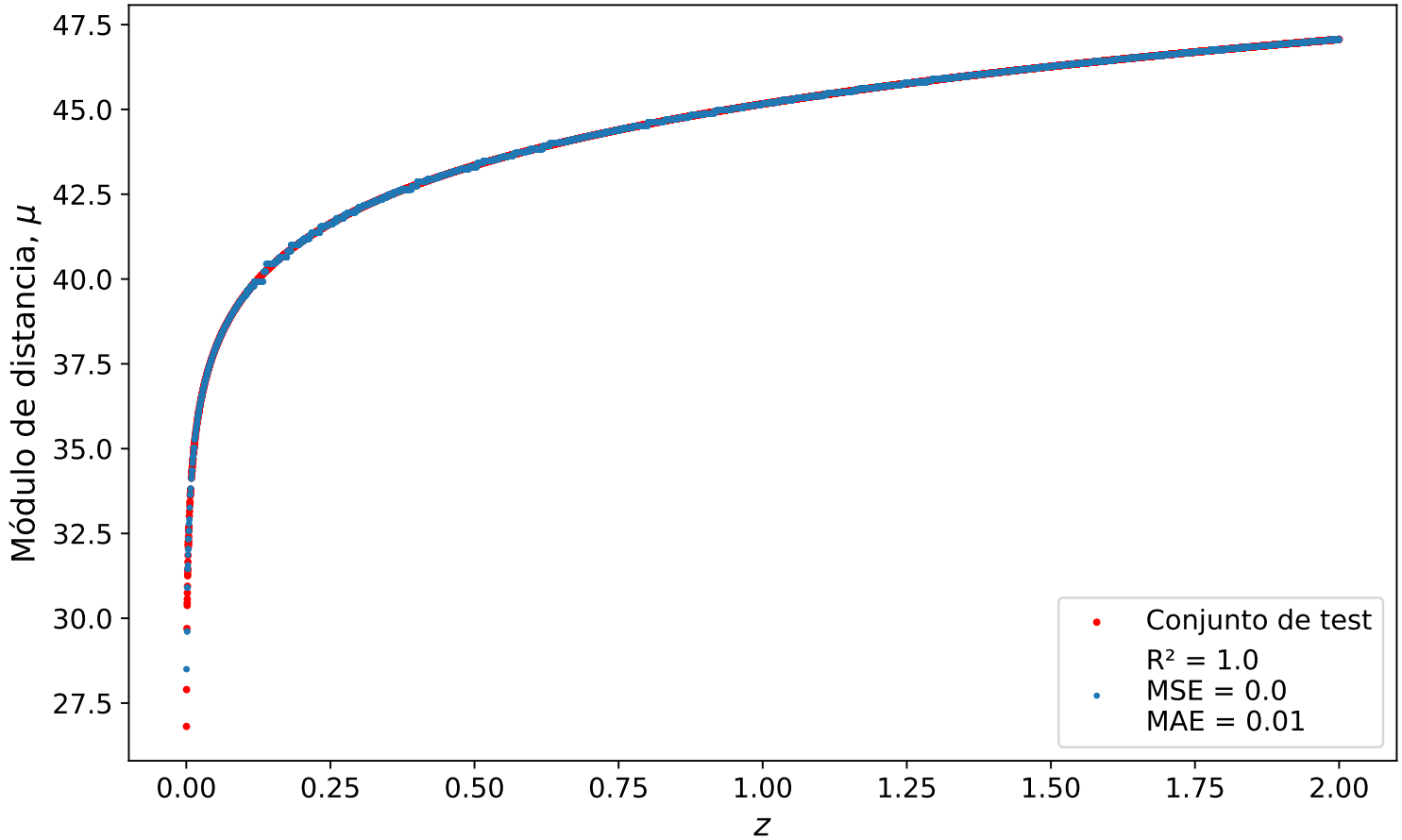
Tiempo de computación

- Tiempo en general 500 puntos, entrenar el modelo y generar 100000 puntos con él.

-	Mathematica	Entrenamiento	Generación 100000 puntos
Tiempo (s)	1,3	0	0

- Tiempo en generar 100000 puntos con Mathematica: 4 minutos 40 segundos.
- Normalmente el rango de z que se usa es mucho menor, por lo que con 500 puntos llenaríamos mejor el intervalo.

Ejemplo más "real".



- Generación de los 500 datos de entrenamiento: aprox. 1 segundo con Mathematica.
- Entrenamiento + Generación de 10000 puntos: instantáneo.

FIN.

¡Muchas gracias por su atención!