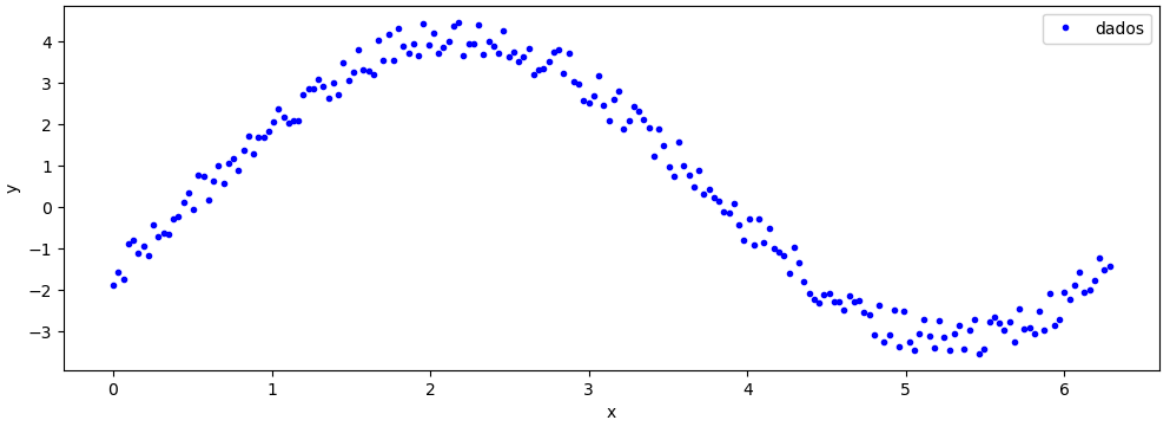
# Laboratório 04 - Ajuste de Curvas via Funções de Base Radial (RBF)

Aluno : Daniel Medeiros Soares Carneiro (20220012347)

**Exemplo 01 :** Sejam $x$, $y$ o seguinte conjunto de pontos:

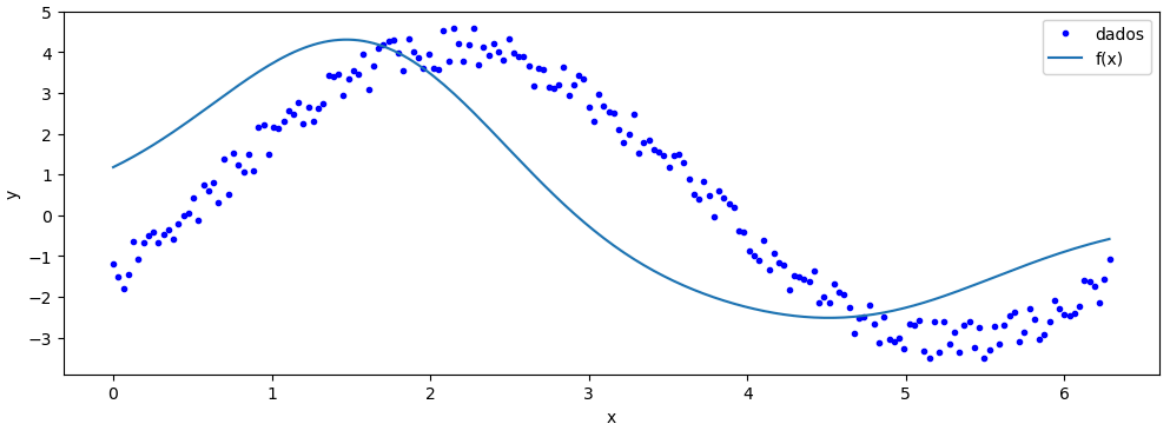
In [ ]:
```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 200)
y = 3*np.sin(x) - 2*np.cos(x) + np.random.random(len(x))
plt.figure(figsize = (12, 4))
plt.plot(x,y,"b.", label = "dados")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```




## Ajuste via Funções de Base Radial (RBF) : Ajuste empírico.

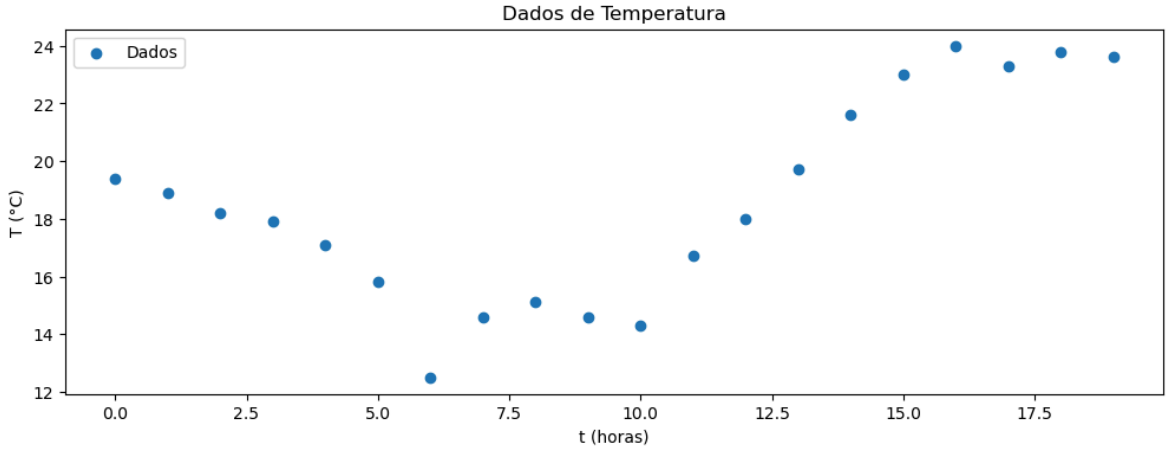
In [4]:
```
c = np.linspace(x[0],x[-1],5)
s2 = .9

phi_0 = np.exp(-(1/(2*s2**2))*(x - c[0])**2)
phi_1 = np.exp(-(1/(2*s2**2))*(x - c[1])**2)
phi_2 = np.exp(-(1/(2*s2**2))*(x - c[2])**2)
phi_3 = np.exp(-(1/(2*s2**2))*(x - c[3])**2)
phi_4 = np.exp(-(1/(2*s2**2))*(x - c[4])**2)

x = np.linspace(0, 2*np.pi, 200)
y = 3*np.sin(x) - 2*np.cos(x) + np.random.random(len(x))
plt.figure(figsize = (12, 4))
plt.plot(x,y,"b.", label = "dados")
#plt.plot(x,-1*phi_0,label = "phi_0")
#plt.plot(x,3.5*phi_1,label = "phi_1")
#plt.plot(x,-3*phi_2,label = "phi_2")
#plt.plot(x,-4*phi_3,label = "phi_3")
plt.plot(x,0.2*phi_0 + 4.5*phi_1 - 1.2*phi_2 - 2.2*phi_3 - 0.1*phi_4,label = "f(
```

```
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```




**Exemplo 02 :** Suponha a medição da temperatura ao longo de 19 horas, representadas pelo seguinte conjunto de dados,

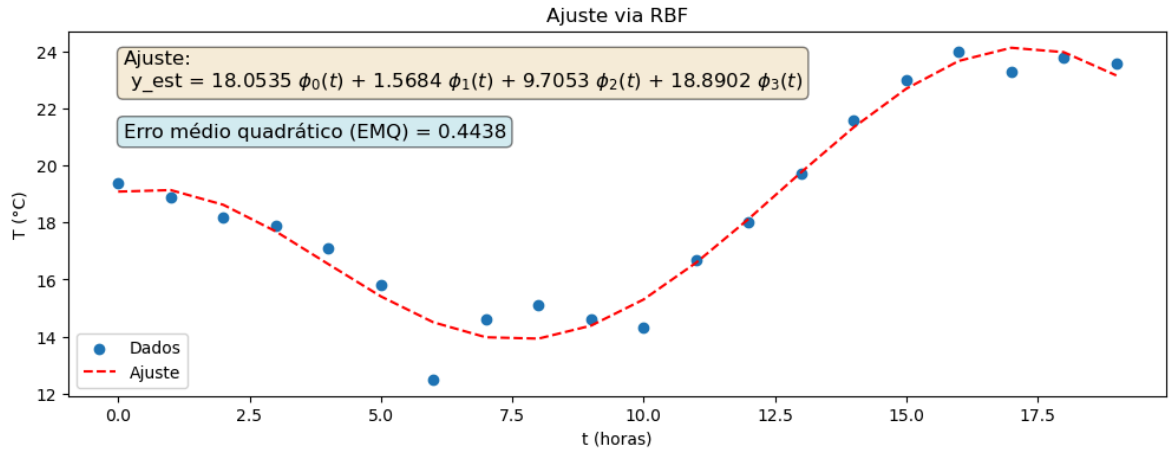
In [6]:
```
t = np.linspace(0,19,20)  # tempo (em horas)
T = np.array([19.4,18.9,18.2,17.9,17.1,15.8,12.5,14.6,15.1,14.6,14.3,
              16.7,18,19.7,21.6,23,24,23.3,23.8,23.6])  # temperatura (em Celsiu

plt.figure(figsize=(12, 4))
plt.scatter(t, T, label='Dados')
plt.xlabel("t (horas)")
plt.ylabel("T (°C)")
plt.title("Dados de Temperatura")
plt.legend()
plt.show()
```




## Ajuste via Funções de Base Radial (RBF) : Resolução de um Sistema de Equações Lineares.

In [8]:
```
c = np.linspace(0, 19, 4)
phi = np.zeros(shape=(len(t), len(c)))

s2 = 4.9

for k in range(len(t)):
    for i in range(len(c)):
```

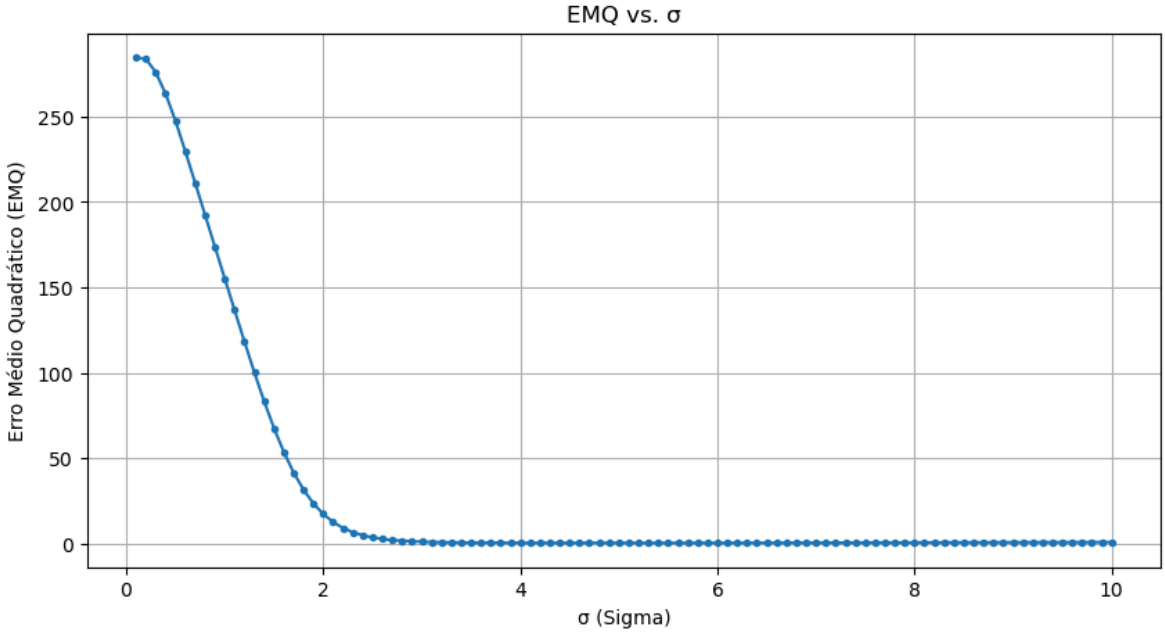
```
        phi[k, i] = np.exp(-(1 / (2 * (s2 ** 2))) * (t[k] - c[i]) ** 2)

w = np.dot(la.pinv(phi), T)

y_est = phi @ w

EMQ = (1 / len(t)) * (sum((T - y_est) ** 2))
#print("O Erro médio quadrático (EMQ) = %.4f" % EMQ)

plt.figure(figsize=(12, 4))
plt.scatter(t, T, label='Dados')
plt.plot(t, y_est, "--r", label='Ajuste')

fit_label = 'Ajuste:\n y_est = %.4f $\\phi_0(t)$ + %.4f $\\phi_1(t)$ + %.4f $\\p
    w[0].item(), w[1].item(), w[2].item(), w[3].item())
plt.text(0.05, 0.95, fit_label, transform=plt.gca().transAxes,
         fontsize=12, verticalalignment='top',
         bbox=dict(boxstyle="round", facecolor='wheat', alpha=0.5))
plt.text(0.05, 0.75, "Erro médio quadrático (EMQ) = %.4f" % EMQ, transform=plt.g
         fontsize=12, verticalalignment='top',
         bbox=dict(boxstyle="round", facecolor='lightblue', alpha=0.5))
plt.xlabel("t (horas)")
plt.ylabel("T (°C)")
plt.title("Ajuste via RBF")
plt.legend()
plt.show()
```




In [9]:

```
c = np.linspace(0, 19, 4)

sigma_range = np.linspace(0.1, 10, 100)
EMQ_values = []

for sigma in sigma_range:
    phi = np.zeros((len(t), len(c)))

    for k in range(len(t)):
        for i in range(len(c)):
            phi[k, i] = np.exp(-(1 / (2 * (sigma ** 2))) * (t[k] - c[i]) ** 2)

    w = la.pinv(phi) @ T
    y_est = phi @ w
    EMQ = (1 / len(t)) * np.sum((T - y_est) ** 2)

    EMQ_values.append(EMQ)

plt.figure(figsize=(10, 5))
```

```
plt.plot(sigma_range, EMQ_values, '-o', markersize=3)
plt.xlabel("σ (Sigma)")
plt.ylabel("Erro Médio Quadrático (EMQ)")
plt.title("EMQ vs. σ")
plt.grid(True)
plt.show()
```




**Atividade 01** - Seja o seguinte conjunto de pontos,

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_k$ | $-1,0$ | $-0,6$ | $-0,2$ | $0,2$ | $0,6$ | $1,0$ |
| $y_k$ | $-0,9602$ | $0,6405$ | $-0,1647$ | $0,34449$ | $-0,3201$ | |
| $c_i$ | $-1,5$ | $-0,5$ | $0,0$ | $0,4$ | $1,0$ | $--$ |

encontre a função analiticamente e, em seguida, implemente-a mostrando os respectivos gráficos e a qualidade do ajuste

$$f(x) = a_0\phi_0(x) + a_1\phi_1(x) + a_2\phi_2(x) + a_3\phi_3(x) + a_4\phi_4(x)$$

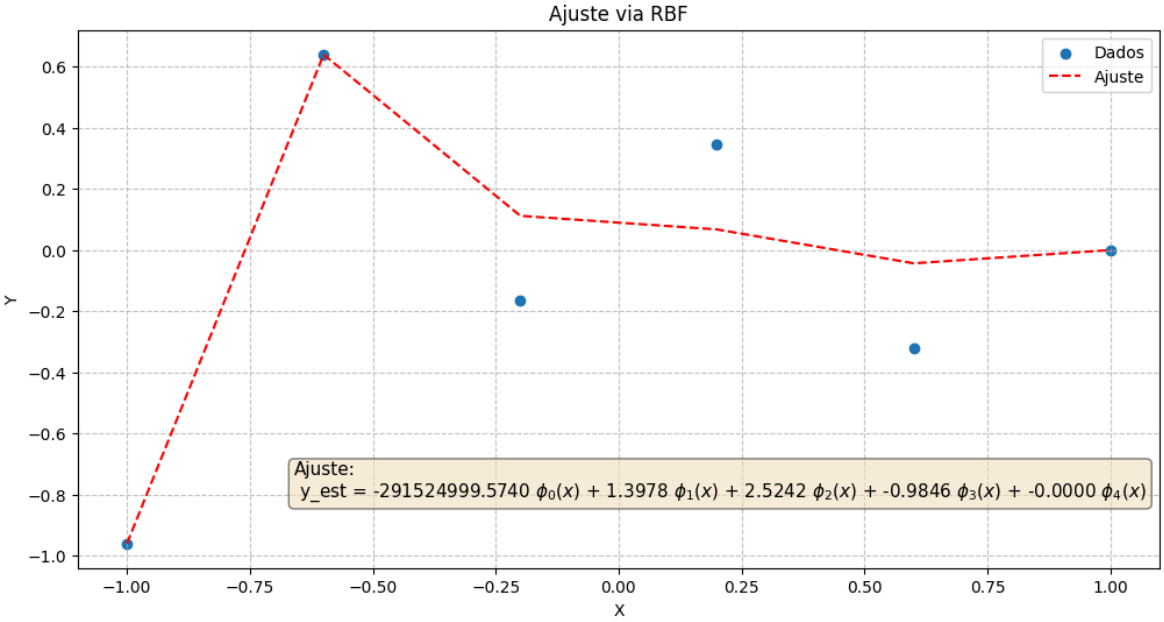
com $\phi_i(x_k) = exp\left(-\left(2\sigma^2\right)^{-1}(x_k - c_i)^2\right)$, $i = 0, 1, \cdots, 4$ (e, $\sigma$ definido pelo usuário) que melhor ajusta o conjunto de pontos. De modo que $EMQ \leq 0,03$.

In [196…

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

sigma = 0.08
x = np.array([-1.0, -0.6, -0.2, 0.2, 0.6, 1.0])
y = np.array([-0.9602, 0.6405, -0.1647, 0.34449, -0.3201, 0.0])
c = np.array([-1.5, -0.5, 0.0, 0.4, 1.0])
phi = np.zeros(shape=(len(x), len(c)))


for k in range(len(x)):
    for i in range(len(c)):
        phi[k, i] = np.exp(-(1 / (2 * sigma**2)) * (x[k] - c[i])**2)
```

```
w = np.dot(la.pinv(phi), y)
y_est = phi @ w

EMQ = (1 / len(x)) * (sum((y - y_est) ** 2))
print(f"Erro Médio Quadrático (EMQ): {EMQ:.5f}")

plt.figure(figsize=(12, 6))
plt.scatter(x, y, label='Dados')
plt.plot(x, y_est, "--r", label='Ajuste')

fit_label = 'Ajuste:\n y_est = %.4f $\\phi_0(x)$ + %.4f $\\phi_1(x)$ + %.4f $\\p
    w[0].item(), w[1].item(), w[2].item(), w[3].item(), w[4].item())
plt.text(0.2, 0.2, fit_label, transform=plt.gca().transAxes,
         fontsize=11, verticalalignment='top',
         bbox=dict(boxstyle="round", facecolor='wheat', alpha=0.5))

plt.xlabel("X")
plt.ylabel("Y")
plt.title("Ajuste via RBF")
plt.grid(True, linestyle="--", alpha=0.7)
plt.legend()
plt.show()
```

```
Erro Médio Quadrático (EMQ): 0.03832
```

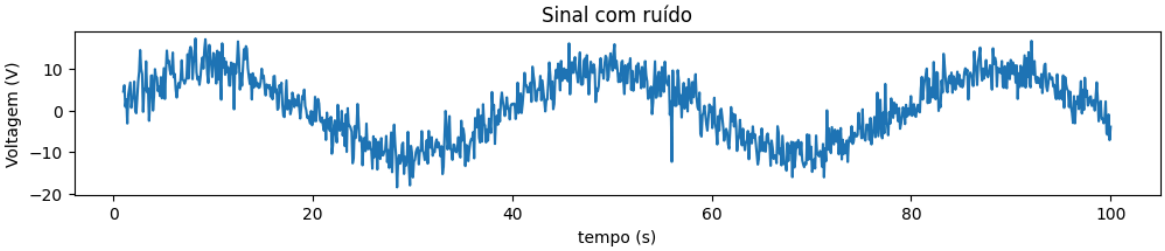



**Atividade 02** - Seja o seguinte conjunto de pontos,

In [203…

```
# Gerando o sinal a ser filtrado
t = np.linspace(1, 100, 1000)
x_volts = 10*np.sin(t/(2*np.pi))

#Gerando o sinal ruidoso e apresentando os gráficos comparativos

t_ruido_db = 10 # Potência do ruído em dB

t_ruido_watts = 10 ** (t_ruido_db / 10) # Convertendo a potência para Watts
x_watts = x_volts ** 2

# Gerando amostras do ruído
media_ruido = 0
```

```
ruido_volts = np.random.normal(media_ruido, np.sqrt(t_ruido_watts), len(x_watts)

y_volts = x_volts + ruido_volts # Sinal acréscido do ruído

plt.figure(figsize = (12, 4))
plt.subplot(212)
plt.plot(t, y_volts) # plot sinal + ruído
plt.title('Sinal com ruído')
plt.ylabel('Voltagem (V)')
plt.xlabel('tempo (s)')
plt.show()
```




Ajuste-o via:

$$f(x) = a_0\phi_0(x) + a_1\phi_1(x) + a_2\phi_2(x) + a_3\phi_3(x) + a_4\phi_4(x)$$

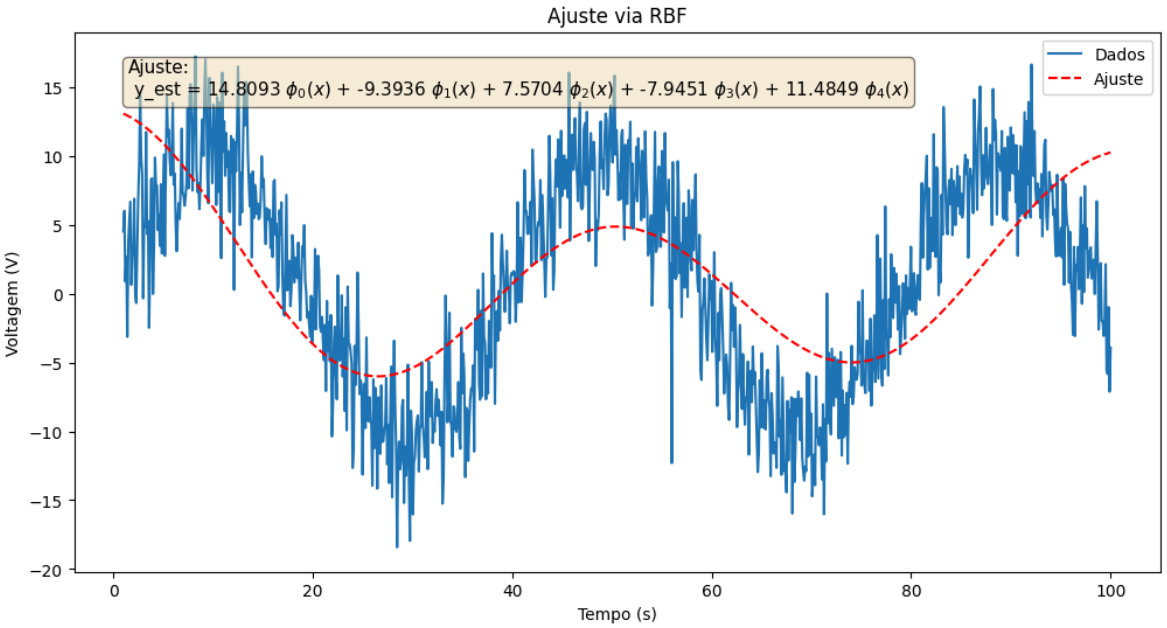
com $\phi_i(x_k) = exp\left(-\left(2\sigma^2\right)^{-1}(x_k - c_i)^2\right)$, $i = 0, 1, \cdots, 4$ (e, $\sigma$ definido pelo usuário) que melhor ajusta o conjunto de pontos.

In [ ]:

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

sigma = 13
c = np.linspace(0, 100, 5)
phi = np.zeros(shape=(len(t), len(c)))

for k in range(len(t)):
    for i in range(len(c)):
        phi[k, i] = np.exp(-(1 / (2 * sigma**2)) * (t[k] - c[i])**2)

w = np.dot(la.pinv(phi), y_volts)

y_est = phi @ w

EMQ = (1 / len(t)) * (sum((y_volts - y_est) ** 2))
print(f"Erro Médio Quadrático (EMQ): {EMQ:.5f}")


plt.figure(figsize=(12, 6))
plt.plot(t, y_volts, label='Dados')
plt.plot(t, y_est, "--r", label='Ajuste')

fit_label = 'Ajuste:\n y_est = %.4f $\\phi_0(t)$ + %.4f $\\phi_1(t)$ + %.4f $\\p
    w[0].item(), w[1].item(), w[2].item(), w[3].item(), w[4].item())

plt.text(0.05, 0.95, fit_label, transform=plt.gca().transAxes,
         fontsize=11, verticalalignment='top',
         bbox=dict(boxstyle="round", facecolor='wheat', alpha=0.5))
```

```
plt.xlabel("Tempo (s)")
plt.ylabel("Voltagem (V)")
plt.title("Ajuste via RBF")
plt.legend()
plt.show()
```

```
Erro Médio Quadrático (EMQ): 32.68936
```

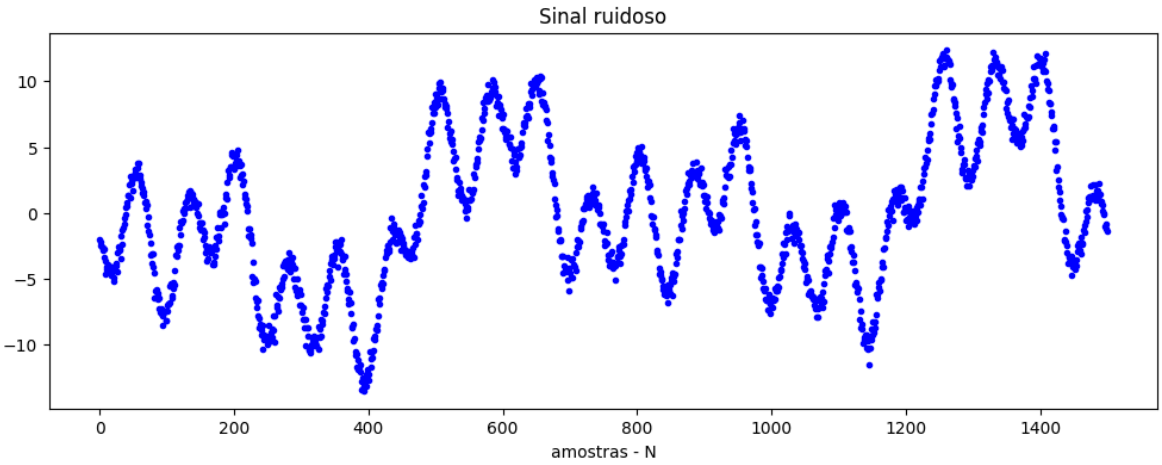



**Atividade 03** - Ajuste o seguinte conjunto de dados via RBF, utilizando

- 500 amostras para o treinamento,
- 500 amostras para o teste e,
- 500 amostras para validação.

In [212…

```
# criando os dados
N = 1500
t = np.linspace([-np.pi, -np.pi/2, -2*np.pi, 0],
[np.pi, np.pi/2, 2*np.pi, 2*np.pi], num = N)
x = np.sin(t) + np.cos(2*t) + np.sin(10*t) #Sinal de entrada
#x = np.sin(np.random.normal(0, 1, (N, 4))) # sinal de entrada
v = np.random.normal(0, 0.5, N) # ruído
d = 2*x[:,0] + 0.1*x[:,1] - 4*x[:,2] + 0.5*x[:,3] + v # desejado + ruído

# Resultados obtidos
plt.figure(figsize=(12,9))
plt.subplot(211);plt.title("Sinal ruidoso");plt.xlabel("amostras - N")
plt.plot(d,"b.", label="d - sinal ruidoso")
plt.show()
```




In [253…

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

indices = np.arange(N)
np.random.shuffle(indices)
train_idx = indices[:500]
test_idx = indices[500:1000]
val_idx = indices[1000:]

X_train, d_train = x[train_idx], d[train_idx]
X_test, d_test = x[test_idx], d[test_idx]
X_val, d_val = x[val_idx], d[val_idx]


n_centroids = 20
centroid_indices = np.random.choice(X_train.shape[0], n_centroids, replace=False
centroids = X_train[centroid_indices]


def rbf_activation(X, centroids, gamma=0.1):
    n_samples = X.shape[0]
    n_centroids = centroids.shape[0]
    activations = np.zeros((n_samples, n_centroids))

    for i in range(n_samples):
        for j in range(n_centroids):
            distance = la.norm(X[i] - centroids[j])
            activations[i, j] = np.exp(-gamma * distance**2)

    return activations


phi_train = rbf_activation(X_train, centroids)
phi_test = rbf_activation(X_test, centroids)
phi_val = rbf_activation(X_val, centroids)


w = la.pinv(phi_train.T @ phi_train) @ phi_train.T @ d_train


def predict(phi, w):
    return phi @ w

d_pred_train = predict(phi_train, w)
```

```
d_pred_test = predict(phi_test, w)
d_pred_val = predict(phi_val, w)


def emq(y_true, y_pred):
    return (1 / len(y_true)) * np.sum((y_true - y_pred) ** 2)

emq_train = emq(d_train, d_pred_train)
emq_test = emq(d_test, d_pred_test)
emq_val = emq(d_val, d_pred_val)

print(f"EMQ Treinamento: {emq_train:.4f}")
print(f"EMQ Teste: {emq_test:.4f}")
print(f"EMQ Validação: {emq_val:.4f}")


plt.figure(figsize=(12, 9))


plt.subplot(2, 1, 1)
plt.title("Ajuste Via RBF")
plt.plot(d_test[:100], "b:", label="Sinal Original com Ruído")
plt.plot(d_pred_test[:100], "r-", label="Sinal Ajustado", alpha=0.7)
plt.xlabel("Amostras")
plt.ylabel("Amplitude")
plt.legend()
plt.tight_layout()
plt.show()
```

```
EMQ Treinamento: 0.5281
EMQ Teste: 0.6215
EMQ Validação: 0.5645
```