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

Solving Systems of Nonlinear Equations

Consider a system of m nonlinear equations of the form

$$\begin{aligned} y_1 &= f_1(\boldsymbol{x}) = 0 \\ y_2 &= f_2(\boldsymbol{x}) = 0 \\ &\vdots \\ y_m &= f_m(\boldsymbol{x}) = 0 \end{aligned}$$

where $\boldsymbol{x} = [x_1, x_2, \ldots, x_m]^T$ is the vector of inputs to the nonlinear equations. We wish to solve for the solution vector, $\boldsymbol{x}$, that satisfies this system of equations. Let $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})]^T$ be the vector of nonlinear equations.

We start be expanding out the Taylor series to a first-order approximation. Let $\bar{\boldsymbol{x}}$ be the fixed point we expand the approximation about.

$$0 = \boldsymbol{f}(\boldsymbol{x}) \overset{\Delta}{=} \boldsymbol{f}(\bar{\boldsymbol{x}}) + \nabla \boldsymbol{f}(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}}) + \mathcal{O}^2$$

$$0 \approx \boldsymbol{f}(\bar{\boldsymbol{x}}) + \nabla \boldsymbol{f}(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}})$$

We omit the higher-order terms and set the approximation equal to zero which will let us solve this problem linearly.

$$0 = \boldsymbol{f}(\bar{\boldsymbol{x}}) + \nabla \boldsymbol{f}(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}})$$

Let's inspect the $\nabla \boldsymbol{f}(\bar{\boldsymbol{x}})$ term. ∇ is the gradiant operator which performs a vector derivative on its operand. Recall that both $\boldsymbol{x}$ and $\boldsymbol{f}(\boldsymbol{x})$ are vector quantities.

$$\nabla \boldsymbol{f}(\bar{\boldsymbol{x}}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\bar{\boldsymbol{x}}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\bar{\boldsymbol{x}}) & \frac{\partial f_1}{\partial x_2}(\bar{\boldsymbol{x}}) & \dots & \frac{\partial f_1}{\partial x_m}(\bar{\boldsymbol{x}}) \\ \frac{\partial f_2}{\partial x_1}(\bar{\boldsymbol{x}}) & \frac{\partial f_2}{\partial x_2}(\bar{\boldsymbol{x}}) & \dots & \frac{\partial f_2}{\partial x_m}(\bar{\boldsymbol{x}}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\bar{\boldsymbol{x}}) & \frac{\partial f_m}{\partial x_2}(\bar{\boldsymbol{x}}) & \dots & \frac{\partial f_m}{\partial x_m}(\bar{\boldsymbol{x}}) \end{bmatrix} = J(\bar{\boldsymbol{x}})$$

Here, we define the Jacobian matrix, $J(\bar{\boldsymbol{x}})$. Each element of the this matrix represents the derivative of one of the nonlinear functions with respect to a specific input evaluated at the linearized point, $\bar{\boldsymbol{x}}$. We can substitute the Jacobian into the first-order approximation.

$$0 = \boldsymbol{f}(\bar{\boldsymbol{x}}) + J(\bar{\boldsymbol{x}})(\boldsymbol{x} - \bar{\boldsymbol{x}})$$

Simplify this expression further by letting $\bar{f} = f(\bar{\boldsymbol{x}})$ and $\bar{J} = J(\bar{\boldsymbol{x}})$.

$$0 = \bar{f} + \bar{J}(\boldsymbol{x} - \bar{\boldsymbol{x}})$$

Rearranging this equation and solving for the solution, $\boldsymbol{x}$, we get

$$\boldsymbol{x} = \bar{\boldsymbol{x}} - \bar{J}^{-1}\bar{f}$$

Recall that $\bar{\boldsymbol{x}}$ representes a fixed-point, $\bar{f} = f(\boldsymbol{x})$, and $\bar{J} = J(\bar{\boldsymbol{x}})$. As an iterative process, we are solving for $\boldsymbol{x}_{k+1}$ given an initial guess, $\boldsymbol{x}_k$. Here we define

$$f_k = f(\boldsymbol{x}_k)$$

$$J_k = J(\boldsymbol{x}_k)$$

We now arrive at the solution to the iterative process for solving systems of nonlinear equations. Note the resemblance to the Newton-Raphson method we learned earlier in the semester which dealt with the one-dimensional (scalar) case. This equation represents the more general form for a system of arbitrary dimensions.

$$\boxed{\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J_k^{-1}\boldsymbol{f}_k}$$

Example 1

Let's try an example. Consider the following system of equations:

$$f_1(\boldsymbol{x}) = x_1^3 + x_2 - 1 = 0$$
$$f_2(\boldsymbol{x}) = -x_1 + x_2^3 + 1 = 0$$

Given the initial guess of $\boldsymbol{x_0} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, find the solution to this systems of nonlinear equations to a residual tolerance of $\|\boldsymbol{f}(\boldsymbol{x})\|_2 < 10^{-8}$. Report the number of iterations, the residual, and the solution.

In [ ]:
```python
# Function
def f(x):
    f1 = x[0]**3 + x[1] - 1
    f2 = -x[0] + x[1]**3 + 1
    fVec = np.array([f1,f2],dtype=float)
    return fVec

# Jacobian
def J(x):
    jacobian = np.array([[3*x[0]**2,1],
                         [-1,3*x[1]**2]],dtype=float)
    return jacobian

# Parameters
x0 = np.array([1,1]) # Initial guess
maxIter = 100 # Max iterations
tol = 1e-8 # Tolerance

# Iterative method
for i in range(1,maxIter+1):
    J_inv = np.linalg.inv(J(x0)) # Inverse Jacobian
    x = x0 - J_inv@f(x0) # Newton-Raphson
    x0 = x

    res = np.linalg.norm(f(x)) # residual

    # Convergence criteria
```

```
    if res < tol:
        print(f'Converged after {i} iterations')
        print(f'Residual = {res}')
        print(f'x = {x}')
        break

    # Failure to converge
    if i == maxIter:
        print(f'Could not converge within {maxIter} Iterations')
```

```
Converged after 6 iterations
Residual = 1.817881472295665e-12
x = [1.00000000e+00 1.82650037e-12]
```

Example 2

Let's try another example. Consider the following system of equations:

$$f_1(\boldsymbol{x}) = 3x_1 + x_1^2 + x_2^2 = 0$$
$$f_2(\boldsymbol{x}) = x_1 x_2 - x_2^2 = 0$$

Given the initial guess of $\boldsymbol{x_0} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$, find the solution to this systems of nonlinear equations to a residual tolerance of $\|\boldsymbol{f}(\boldsymbol{x})\|_2 < 10^{-10}$. Report the number of iterations, residual, and the solution.

In [ ]:
```
# Function
def f(x):
    f1 = 3*x[0] + x[0]**2 + x[1]**2
    f2 = x[0]*x[1] - x[1]**2
    fVec = np.array([f1,f2])
    return fVec

# Jacobian
def J(x):
    jacobian = np.array([[3+2*x[0],2*x[1]],
                         [x[1],2*x[0]-2*x[1]]],dtype=float)
    return jacobian

# Parameters
x0 = np.array([2,2]) # Initial guess
```

```
maxIter = 100 # Max iterations
tol = 1e-10 # Tolerance

# Iterative method
for i in range(1,maxIter+1):
    J_inv = np.linalg.inv(J(x0)) # Inverse Jacobian
    x = x0 - J_inv@f(x0) # Newton-Raphson
    x0 = x

    res = np.linalg.norm(f(x)) # residual

    # Convergence criteria
    if res < tol:
        print(f'Converged after {i} iterations')
        print(f'Residual = {res}')
        print(f'x = {x}')
        break

    # Failure to converge
    if i == maxIter:
        print(f'Could not converge within {maxIter} Iterations')
```

```
Converged after 20 iterations
Residual = 3.7313554768583766e-11
x = [ 1.17618314e-26 -5.13660078e-06]
```

Nonlinear Least-Squares Iterative Method

Let $n > m$. For a given set of n data points, $[x_n, y_n]$,

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

a set of m unknown parameters, $\boldsymbol{c}$,

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}$$

and a nonlinear fitting function, $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{c})$, we want to find the solution that minimizes the L_2-norm of the residual, $\boldsymbol{r} = \hat{\boldsymbol{y}} - \boldsymbol{y}$. The least-squares minimization problem is given by

$$\min \sum_{k=1}^{n} \left(\hat{y}(x_k) - y_k\right)^2$$

Unlike the classic linear least-squares problem, we cannot immediately convert the equation $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{c})$ into a linear form $\boldsymbol{y} = A\boldsymbol{x}$ since the parameters are embedded in the equation in a nonlinear fashion that cannot be simplified. To remedy this, we will start by considering the residual equation which is equal to zero in the most ideal case (we want to minimize it). Recall how we defined the residual

$$\boldsymbol{r} = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

More explicitly, we write

$$\boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) = f(\boldsymbol{x}, \boldsymbol{c}) - \boldsymbol{y} = 0$$

Now we can linearize the residual function, $\boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c})$, about a fixed-point, $\bar{\boldsymbol{c}}$, by means of a first-order Taylor series expansion.

$$0 = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}) \overset{\Delta}{=} \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}}) + \nabla \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}})(\boldsymbol{c} - \bar{\boldsymbol{c}}) + \mathcal{O}^2$$

$$0 \approx \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}}) + \nabla \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}})(\boldsymbol{c} - \bar{\boldsymbol{c}})$$

We omit the higher-order terms and set this approximation equal to zero which will let us solve this problem linearly.

$$0 = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}}) + \nabla \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}})(\boldsymbol{c} - \bar{\boldsymbol{c}})$$

Let's quickly inspect the gradient of the residual to see how we can simplify the expression.

$$\begin{aligned} \nabla \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}}) &= \nabla \left(f(\boldsymbol{x}, \bar{\boldsymbol{c}}) - \boldsymbol{y}\right) \\ &= \nabla f(\boldsymbol{x}, \bar{\boldsymbol{c}}) - \nabla \boldsymbol{y} \\ &= \nabla f(\boldsymbol{x}, \bar{\boldsymbol{c}}) \end{aligned}$$

Therefore, we can rewrite the linear approximation as

$$0 = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}}) + \nabla f(\boldsymbol{x}, \bar{\boldsymbol{c}})(\boldsymbol{c} - \bar{\boldsymbol{c}})$$

Similar to our earlier discussion on solving systems of nonlinear equations, $\nabla f(\boldsymbol{x}, \bar{\boldsymbol{c}})$ will return a Jacobian matrix. Note that for this problem, the unknown parameters are in the vector $\boldsymbol{c}$ (not $\boldsymbol{x}$, which contains the input data points). This implies that our gradient is with respect to the coefficients, $\boldsymbol{c}$. In other words, $\nabla = \dfrac{\partial}{\partial \boldsymbol{c}}$. In the case of the nonlinear least-squares problem, the Jacobian matrix will have the form,

$$\nabla f(\boldsymbol{x}, \bar{\boldsymbol{c}}) = \frac{\partial f}{\partial \boldsymbol{c}}(\boldsymbol{x}, \bar{\boldsymbol{c}}) = \begin{bmatrix} \dfrac{\partial f}{\partial c_1}(x_1, \bar{\boldsymbol{c}}) & \dfrac{\partial f}{\partial c_2}(x_1, \bar{\boldsymbol{c}}) & \dots & \dfrac{\partial f}{\partial c_m}(x_1, \bar{\boldsymbol{c}}) \\ \dfrac{\partial f}{\partial c_1}(x_2, \bar{\boldsymbol{c}}) & \dfrac{\partial f}{\partial c_2}(x_2, \bar{\boldsymbol{c}}) & \dots & \dfrac{\partial f}{\partial c_m}(x_2, \bar{\boldsymbol{c}}) \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f}{\partial c_1}(x_n, \bar{\boldsymbol{c}}) & \dfrac{\partial f}{\partial c_2}(x_n, \bar{\boldsymbol{c}}) & \dots & \dfrac{\partial f}{\partial c_m}(x_n, \bar{\boldsymbol{c}}) \end{bmatrix} = J(\boldsymbol{x}, \bar{\boldsymbol{c}})$$

Note that the rows correspond to the n data points, $\boldsymbol{x}$, and the columns correspond to the partial derivative of the fitting function with respect to the m unknown parameters, $\boldsymbol{c}$. We can substitute the Jacobian into the first order approximation and let $\Delta \boldsymbol{c} = (\boldsymbol{c} - \bar{\boldsymbol{c}})$ to get

$$0 = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}}) + J(\boldsymbol{x}, \bar{\boldsymbol{c}})\Delta \boldsymbol{c}$$

Simplify this expression further by letting $\bar{\boldsymbol{r}} = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}})$ and $\bar{J} = J(\boldsymbol{x}, \bar{\boldsymbol{c}})$

$$0 = \bar{\boldsymbol{r}} + \bar{J}\,\Delta \boldsymbol{c}$$

Recall that the vector $\boldsymbol{c}$ contains the unknown paramters we wish to solve for. It is embedded in the term $\Delta \boldsymbol{c} = (\boldsymbol{c} - \bar{\boldsymbol{c}})$. Therefore, if we can solve $\Delta \boldsymbol{c}$, we can extract the solution $\boldsymbol{c}$. Rearrange this equation to get it into a form that can be solved with the linear-least squares solution.

$$\boxed{-\bar{\boldsymbol{r}} = \bar{J}\,\Delta \boldsymbol{c}} \implies \boldsymbol{y} = A\boldsymbol{x}$$

From previous lessons, we already know the solution to this problem so we can solve for the difference vector, $\Delta \boldsymbol{c}$, with the equation

$$\Delta \boldsymbol{c} = -\left(\bar{J}^T \bar{J}\right)^{-1} \bar{J}^T \bar{\boldsymbol{r}}$$

Substituting for $\Delta \boldsymbol{c} = (\boldsymbol{c} - \bar{\boldsymbol{c}})$ and rearranging the equation, solve for the unknown parameters, $\boldsymbol{c}$.

$$\boldsymbol{c} - \bar{\boldsymbol{c}} = -\left(\bar{J}^T \bar{J}\right)^{-1} \bar{J}^T \bar{\boldsymbol{r}}$$

$$\boldsymbol{c} = \bar{\boldsymbol{c}} - \left(\bar{J}^T \bar{J}\right)^{-1} \bar{J}^T \bar{\boldsymbol{r}}$$

Recall that $\bar{\boldsymbol{c}}$ representes a fixed-point, $\bar{J} = J(\boldsymbol{x}, \bar{\boldsymbol{c}})$, and $\bar{\boldsymbol{r}} = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \bar{\boldsymbol{c}})$. As an iterative process, we are solving for $\boldsymbol{c}_{k+1}$ given an initial guess, $\boldsymbol{c}_k$. Here we define

$$J_k = J(\boldsymbol{x}, \boldsymbol{c}_k)$$

$$\boldsymbol{r}_k = \boldsymbol{r}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{c}_k)$$

We now arrive at the solution to the iterative nonlinear least-squares problem

$$\boxed{\boldsymbol{c}_{k+1} = \boldsymbol{c}_k - \left(J_k^T J_k\right)^{-1} J_k^T \boldsymbol{r}_k}$$

This method converges under the following error and residual criteria

$$\begin{aligned} \text{Error Criteria: } \|\boldsymbol{c}_{k+1} - \boldsymbol{c}_k\| < \|\boldsymbol{c}_k - \boldsymbol{c}_{k-1}\| &\implies \|\Delta \boldsymbol{c}_{k+1}\| < \|\Delta \boldsymbol{c}_k\| \\ \text{Residual Criteria: } \|\boldsymbol{r}_{k+1} - \boldsymbol{r}_k\| < \|\boldsymbol{r}_k - \boldsymbol{r}_{k-1}\| &\implies \|\Delta \boldsymbol{r}_{k+1}\| < \|\Delta \boldsymbol{r}_k\| \end{aligned}$$

where $\boldsymbol{f}_k = f(\boldsymbol{x}, \boldsymbol{c}_k)$. When asked to iterate this method to a specified tolerance, ε, this criteria is expressed as

$$\begin{aligned} \text{Error Criteria: } \|\boldsymbol{c}_{k+1} - \boldsymbol{c}_k\| < \varepsilon \\ \text{Residual Criteria: } \|\boldsymbol{r}_{k+1} - \boldsymbol{r}_k\| < \varepsilon \end{aligned}$$

Example

Create 100 data points for x uniformly distributed between -1.5 and 1.5 with the equation

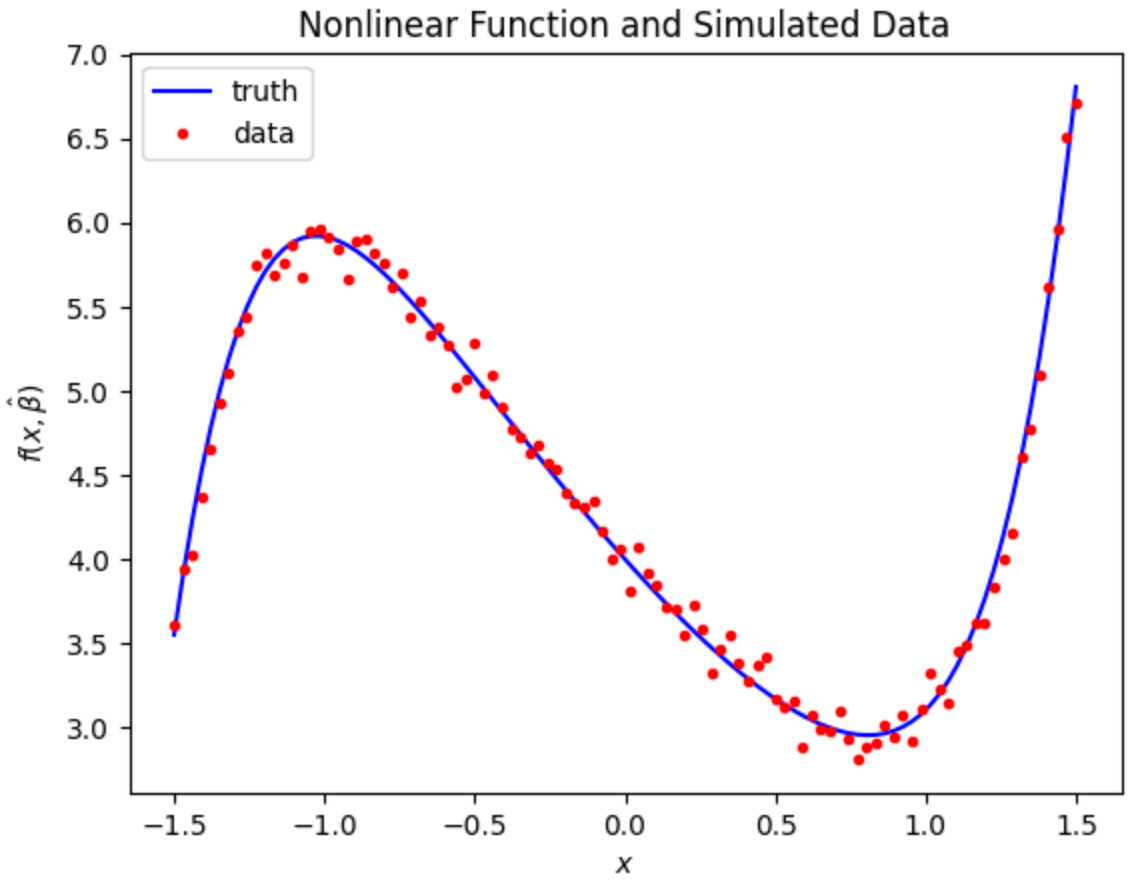
$$f(\boldsymbol{x},\boldsymbol{\beta}) = \beta_1 x^3 + \beta_2 e^{\beta_3 x} + \beta_4 x \sin x^2$$

where

$$\boldsymbol{\beta}_{true} = \begin{bmatrix} 2 \\ 4 \\ -0.5 \\ -\pi/2 \end{bmatrix}$$

Corrupt the measurments with Gaussian noise $\mathcal{N}(0, 0.1)$. Estimate the parameters by iterative nonlinear least-squares with initial guess $\bar{\boldsymbol{\beta}} = [1, 1, 1, 1]^T$ to an L_2-normed residual tolerance of 1×10^{-5}

In [ ]:
```
# Function
def f(x,B):
    return B[0]*x**3 + B[1]*np.exp(B[2]*x) + B[3]*x*np.sin(x**2)

# Jacobian
def J(x,B):
    return np.array([x**3, np.exp(B[2]*x), x*B[1]*np.exp(B[2]*x), x*np.sin(x**2)]).T

# True parameters
B_true = np.array([2,4,-0.5,-np.pi/2])

# x-values
x = np.linspace(-1.5,1.5,100)

# Noise and measurement
noise = np.random.normal(0,0.1,100)
f_true =  f(x,B_true)
f_meas =  f_true + noise

# Plot
plt.plot(x,f_true,'b-',label='truth')
plt.plot(x,f_meas,'r.',label='data')
plt.xlabel(r'$x$')
plt.ylabel(r'$f(x,\hat{\beta})$')
plt.title('Nonlinear Function and Simulated Data')
plt.legend()
plt.show()
```



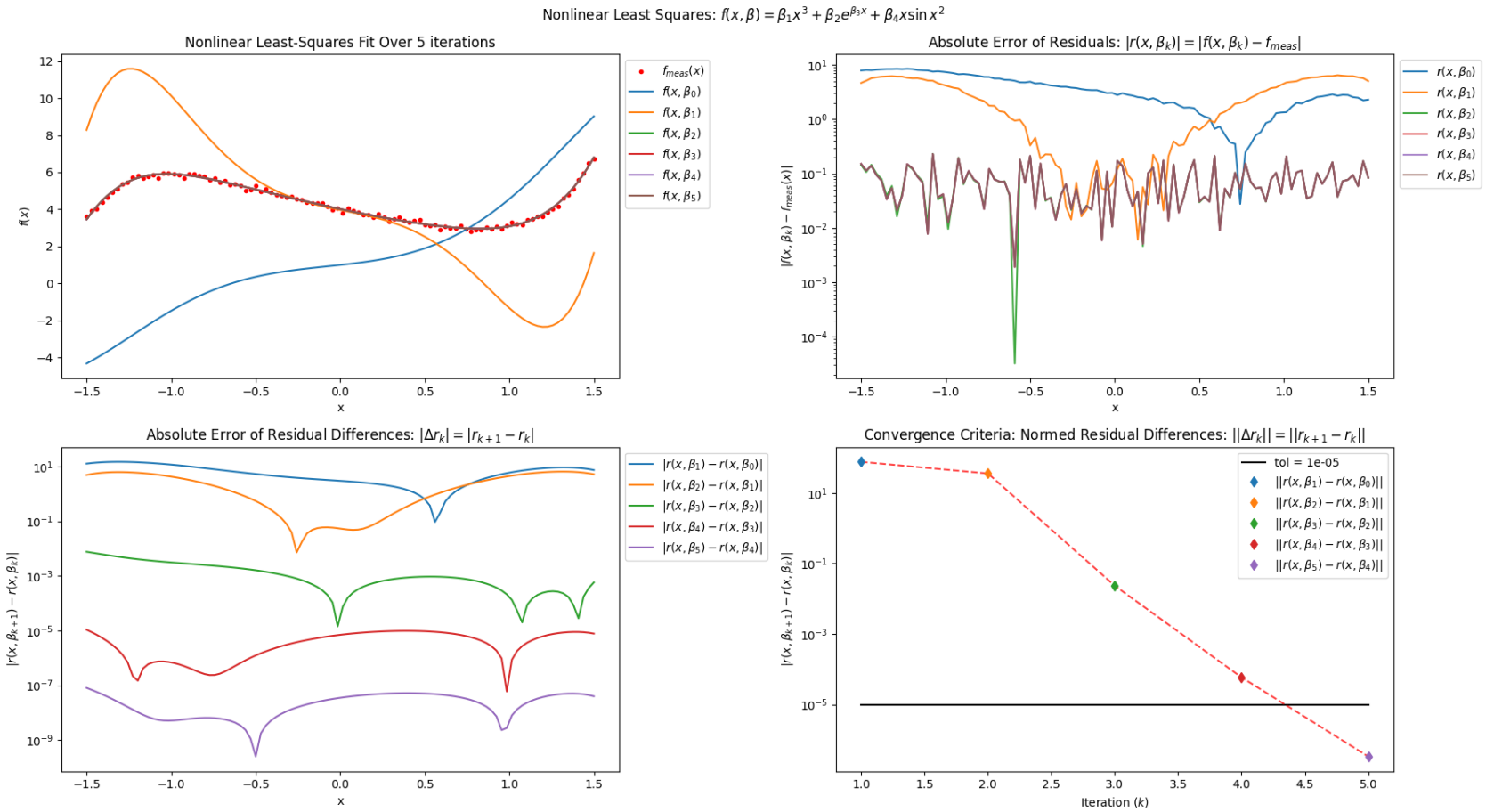
In [ ]:
```
# Given parameters
B_0 = np.array([1,1,1,1],dtype=float) # Initial guess
tol = 1e-5 # Tolerance
maxIter = 100 # Max iteration

# Initial residual
res0 = f(x,B_0) - f_meas

# Lost history of values for plotting
B_hist = np.array([B_0.copy()]) # Parameter history (unknown coefficients)
res_hist = np.array([abs(res0.copy())]) # Residual history
conv_hist = np.array([abs(res0.copy())]) # Convergence criteria history

# Iterate for loop until convergence criteria is met
```

```
for i in range(1,maxIter+1):
    B = B_0 - np.linalg.pinv(J(x,B_0)) @ res0 # Iterative nonlinear least-squares
    res1 = f(x,B) - f_meas # New residual
    convCrit = abs(res0 - res1) # Convergence criteria: vector of residual differences = |res(k) - res(k+1)|

    # Append history
    B_hist = np.concatenate([B_hist,[B]],0) # Append current parameters
    res_hist = np.concatenate([res_hist,[abs(res1)]],0) # Append current residual
    conv_hist = np.concatenate([conv_hist,[convCrit]],0) # Append current convergence criteria (as a vector)

    # Reset variables for next iteration
    B_0 = B
    res0 = res1

    # Normed difference of residual vectors = ||res(k) - res(k+1)||
    convCritNorm = np.linalg.norm(convCrit)
    if convCritNorm < tol:
        print(f'Converged after {i} iterations')
        print(f'Normed residual difference = {convCritNorm}')
        print(f'B_final = {B}')
        break

    # Failure to converge
    if i == maxIter:
        print(f'Could not converge within {maxIter} Iterations')

# First row is a placeholder
conv_hist = np.delete(conv_hist,0,0)

# Absolute error and normed error
err = abs(B - B_true)
errNorm = np.linalg.norm(err)
print(f'Final error = {err}')
print(f'Normed error = {errNorm}')
```

```
Converged after 5 iterations
Normed residual difference = 3.3139443392732097e-07
B_final = [ 2.01735458  4.01768931 -0.48469073 -1.67069623]
Final error = [0.01735458 0.01768931 0.01530927 0.0998999 ]
Normed error = 0.10405987000549706
```

In [ ]:
```
# Plot figures
fig, ax = plt.subplots(2,2, figsize=(18,10)) # Create subplot figure

ax[0][0].plot(x,f_meas,'r.',label=r'$f_{meas}(x)$') # Measured data points

# Plot results over each iteration
iter = len(B_hist)-1
for i in range(0,iter+1):
    ax[0][0].plot(x,f(x,B_hist[i]),label=rf'$f(x,\beta_{i})$') # Nonlinear fit
    ax[0][1].semilogy(x,res_hist[i,:],label=rf'$r(x,\beta_{i})$',alpha=1-0.05*i) # Residual = |estimate - true|
    if i > 0:
        ax[1][0].semilogy(x,conv_hist[i-1,:],label=rf'$\left|r(x,\beta_{i}) - r(x,\beta_{i-1}) \right|$') # Absolute

# Visualize tolerance
ax[1][1].semilogy(np.arange(1,iter+1),tol*np.ones(iter),'k-',label=f'tol = {tol}')
# Separate loop to format labels in legend
ax[1][1].semilogy(np.arange(1,iter+1),np.linalg.norm(conv_hist,2,1),'r--',alpha=0.75)
for i in range(0,iter):
    # Normed difference of residuals ||res(k) - res(k+1)||
    ax[1][1].semilogy(i+1,np.linalg.norm(conv_hist[i,:]),'d',label=rf'$\left||r(x,\beta_{i+1}) - r(x,\beta_{i}) \righ



# Parameters for 1st plot
ax[0][0].set_xlabel('x')
ax[0][0].set_ylabel(r'$f(x)$')
ax[0][0].set_title(f'Nonlinear Least-Squares Fit Over {iter} iterations')
ax[0][0].legend(bbox_to_anchor=(1, 1))

# Parameters for 2nd plot
ax[0][1].set_xlabel('x')
ax[0][1].set_ylabel(r'$\left|f(x,\beta_{k}) - f_{meas}(x) \right|$')
ax[0][1].set_title(r'Absolute Error of Residuals: $|r(x,\beta_k)| = |f(x,\beta_k) - f_{meas}|$')
ax[0][1].legend(bbox_to_anchor=(1, 1))

# Parameters for 3rd plot
ax[1][0].set_xlabel('x')
ax[1][0].set_ylabel(r'$\left|r(x,\beta_{k+1}) - r(x,\beta_{k}) \right|$')
ax[1][0].set_title(r'Absolute Error of Residual Differences: $|\Delta r_k| = |r_{k+1} - r_k|$')
ax[1][0].legend(bbox_to_anchor=(1, 1))

# Parameters for 4th plot
```

```
ax[1][1].set_xlabel(r'Iteration ($k$)')
ax[1][1].set_ylabel(r'$\left|r(x,\beta_{k+1}) - r(x,\beta_{k}) \right|$')
ax[1][1].set_title(r'Convergence Criteria: Normed Residual Differences: $||\Delta r_k|| = ||r_{k+1} - r_k||$')
ax[1][1].legend(bbox_to_anchor=(1, 1))

fig.suptitle(r'Nonlinear Least Squares: $f(x,\beta) = \beta_1 x^3 + \beta_2 e^{\beta_3 x} + \beta_4 x \sin{x^2}$')
plt.tight_layout()
plt.show()
```



Break Down of the Example

Top Left Plot: Fitting Curve Evolution

This problem deals with solving for the coefficients of the function that will minimize the L_2-normed residual of the curve. Basically, we are solving for the coeffieicents the make the function fit the data. The blue curve corresponts to the function with the coefficients given in the initial guess. As we iterate through this nonlinear least-squares method, we find coefficients that more and more accurately fit the curve. The last three curves are so close together, it is visually difficult to discern them here.

Top Right Plot: Residual Evolution

Here we are wanting to vizualize how the absolute value of the residual changes with each iteration. Recall that the residual is the difference between the fitting curve and the measurements or data points. We expect the residuals to get smaller as we iterate, but does it ever actually apporach zero? Think about what this means. If the residual equals zero, that means that the curve has successfully hit all of the data points. This is simply not possible with curve fitting because our data points will realistically never lie exactly on the curve due to noisy measurements. Therefore, instead of converging to zero, we expect the residual to converge to some non-zero value, which is exactly what we see in this plot. True, the curve is getting more accurate with each iteration, but it will not converge to zero. How then, can we demonstrate that our fitting curve is converging to the correct coefficient values? We'll see this in the last two plots.

Bottom Left Plot: Residual Difference Between Iterations

Even though the residals will not converge to zero, they will still converge to some non-zero value. This means that we can still expect the most recently computed residual to be smaller than the value from the previous iteration. We can visualize this by plotting the absolute difference between the residuals from two consecutive iterations. As we can see from the figure, this is indeed what is happening.

Bottom Right Plot: Convergence Criteria and Normed Residual Differences

The previous plot is great for showing the convergence of this method, but how do we know when to stop converging? This problem explicitly tells us to stop when the normed difference between consecutive residuals is below a set tolerance. However, this criteria could have just as easily been when the normed difference between consecutive coefficients is below a tolerance. Again, we see that the convergence criteria is a design parameter that can be set by a designer or can be provided for you. This figure takes each curve from the previous plot and computes the L_2-norm of the curve, plotting this value for each iteration. We also see the tolerance that was set as a solid black line. Once we cross under this line, we stop iterating because we have achieved the desired minimum tolerance.