Curve Fitting using reduced QR Decomposition

Let $A \in \mathbb{R}^{m\times n}$ be a matrix with $m \geq n$ and linearly independent columns and let $b \in \mathbb{R}^m$. Solving Least Squares Problems of the form

$$\min_{x\in\mathbb{R}^n} \|Ax - b\|^2$$

is equivalent to solving the so-called *normal equation* (derivation follows later in the lecture)

$$A^\top Ax = A^\top b.$$

This is a linear system and we can apply a "factor and solve" approach. Specifically, assume we have the reduced $\widehat{Q}\widehat{R}$, where $\widehat{R}$ is invertible since A has full column rank by assumption. Now, inserting $A = \widehat{Q}\widehat{R}$ into the normal equation gives

$$A^T Ax = A^T b \;\Leftrightarrow\; \widehat{R}^T\widehat{R}x = \widehat{R}^T\widehat{Q}^T b \;\Leftrightarrow\; \widehat{R}x = \widehat{Q}^T b.$$

Thus, we can finally compute a least squares solution by solving a triangular system

$$\widehat{R}x = \widehat{Q}^T b.$$

Interesting Observation: Applying a "factor and solve" approach to $Ax = b$ results in the same the systems!

Task

Assume you were given some numpy.ndarray called "data" containing measurements $(z_1, y_1), \ldots, (z_m, y_m) \in \mathbb{R}\times\mathbb{R}$. Further assume that this data has shape `(2,m)` so that the first row `data[0,:]` contains the z_i values and the second row `data[1,:]` contains the y_i values.

1. Implement a function `poly_curve_fit(data, p)` which computes a polynomial fit to the data.

   The **input** `data` shall have the form as described above and `p` shall determine the polynomial used to fit the data. More precisely, `p` shall be a list `[p1, ..., pn]` containing $n \leq m$ *distinct* natural numbers between 0 and $m-1$ which determine the polynomial model f by

   $$f(z) = c_1 z^{p_1} + c_2 z^{p_2} + \ldots + c_k z^{p_n} = \sum_{j=1}^{n} c_j f_j(z) \quad \text{with} \quad f_j(z) := z^{p_j}.$$

   With other words, the input `p` determines which *monomials* z^p we want to use for our model. For example, `p = [0,1]` amounts for a linear model

   $$f(z) = c_1 z^{p_1} + c_2 z^{p_2} = c_1 z^0 + c_2 z^1 = c_1 + c_2 z^1.$$

   Then the **function** `poly_curve_fit(data, p)` shall determine appropriate coefficients $c_1, \ldots, c_n$ by following this recipe:

   a) Assemble the vector $b = (y_1, \ldots, y_m) \in \mathbb{R}^m$.
   b) Assemble the matrix $A = (a_{ij})_{ij} \in \mathbb{R}^{m\times n}$, where $a_{ij} := f_j(z_i)$.
   c) Solve the least squares problem $\min_x \|Ax - b\|^2$ via the normal equation above by using the functions `qr_factor(A)` and `solve_tri((Q,R),b)` from previous exercises (or appropriate SciPy routines).
      Remark: One can show that the columns of A are independent if the z_i are *distinct* (also see *Vandermonde matrix*)!
   d) Plot the measurements and the fitting polynomial into one figure.

   The function shall **output** the solution parameters $x = (c_1, \ldots, c_n)$.

2. Test your routine by fitting the data

z	-2	-1	0	1	2
y	-2	1	-1	2	6

   with different polynomials of your choice.

3. Find a Scipy routine to solve the least squares problem $\min_x \|Ax - b\|^2$. If you want, you can extend the parameter interface `poly_curve_fit(data, p, own=True)` by an optional parameter, for example own=True, which you can use to switch between either our approach with a (reduced) QR-decomposition or a SciPy routine to solve the least squares problem.

Solution:

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

# the given data containing 5 measurements
data = np.array([[-2.,-1., 0., 1., 2.],
                 [-2., 1., -1., 2., 6.]])
#data = np.array([[-1., 0., 1.],
#                  [-1., 3., 1.]])

# polynomial curve fitting
def poly_curve_fit(data, p, own=True):
    """
    INPUT:
        numpy.ndarray data of shape (2,m) with
            data[0,:] = (z_1, ..., z_m) explanatory/independent variables
            data[1,:] = (y_1, ..., y_m) response/dependent variables
        list p = [p1, p2,..., pn] determining the polynomial model
            f_c(z) = c1 * z^p1 + ... + cn * z^pn
        own : switch to use either our approach with (reduced) QR-decomposition
              or SciPy's routine to solve the least squares problem

    OUTPUT:
        numpy.ndarray c of shape (n,) such that
           c = argmin_c sum_i (f_c(z_i) - y_i)^2

    """
    # (a) assemble the vector b
    b = data[1,:]

    # (b) assemble the matrix A
    z_i = data[0,:][np.newaxis].T
    A = z_i**p

    # (c) determine c by solving using QR Decomposition
    if own==True:
        # "factor_qr"
        import scipy.linalg as linalg
        Q, R = linalg.qr(A)
        m, n = len(b), len(p)
        Q, R = Q[0:m,0:n], R[0:n,0:n]
        # "solve_qr"
        c = linalg.solve_triangular(R, Q.T @ b )
    else:
        import scipy.linalg as linalg
        c , res , rnk , s = linalg.lstsq(A,b)

    # (d) Plot measurements and fitting polynomial into one figure
    print("\n--------------------------\nExample: p =", p, "\nown =", own )
    Z = np.linspace(-2, 2, 50) # other z values
    Y = (Z[np.newaxis].T**p).dot(c) # evaluate model on Z
    plt.figure()
```

```
    plt.title("Polynomial degree = " + str(max(p)))
    plt.plot(z_i, b, 'ro')
    plt.plot(Z, Y, 'b')
    plt.show()

    return c

if __name__ == "__main__":
    # different choices of polynomials
    P = [[1], [0,1], [0,1,2], [0, 1, 2, 3], [0, 1, 2 ,3, 4] ]

    # apply the routine for all those choices
    for p in P:
        for own in [True, "Scipy <lstsq>"]:
            poly_curve_fit(data, p, own = own)
        print("\n")
```