S11 delay and polyfitting after S11 calibaration

Titu Samson

1 Introduction

The calibration of the sky temperature, T_{sky}, is a crucial step in the analysis of radio astronomical data. The process involves reading calibration data files, fitting S-parameters, and using these fits to calculate T_{sky}.

2 Theory of Delay Calculation

In the calibration and fitting of the S_{11} parameter, it is often necessary to account for delays introduced in the measurement setup. This section outlines the theory behind calculating the optimal delay to align the phase of the measured S_{11} data.

2.1 Weighted Sum of S_{11}

Given:

- $S_{11,i}$: The S_{11} parameter measured at the i-th frequency.
- w_i: The weight associated with the i-th frequency (now the weight is an array of ones, i.e., no weight).
- f_i: The i-th frequency.
- τ: The delay being tested.
- n: The total number of frequency points.

The weighted sum with a delay τ is calculated as:

$$\text{Sum}(\tau) = \sum_{i=0}^{n-1} w_i S_{11,i} e^{j2\pi f_i \tau} \tag{1}$$

2.2 Finding the Optimal Delay

The goal is to find the delay τ that maximizes the magnitude of this weighted sum:

$$\max_{\tau} |\text{Sum}(\tau)| \tag{2}$$

To achieve this, the algorithm iterates over a range of potential delays and calculates the weighted sum for each delay. The delay that results in the maximum magnitude of the weighted sum is chosen as the optimal delay.

2.3 Mathematical Steps

1. Initialize Parameters: Set the initial maximum value to a very low number and initialize the best delay.

$$\text{max\_val} = -1 \times 10^{99} \tag{3}$$

$$\text{best\_delay} = 0 \tag{4}$$

2. Iterate Over Delays: For each potential delay τ, calculate the weighted sum:

$$\text{sumc} = \sum_{i=0}^{n-1} w_i S_{11,i} e^{j2\pi f_i \tau} \tag{5}$$

3. Evaluate Magnitude: Compute the magnitude of the sum:

$$\text{magnitude} = |\text{sumc}| \tag{6}$$

4. Update Optimal Delay: If the magnitude is greater than the current maximum value, update the maximum value and best delay:

$$\text{if magnitude} > \text{max\_val} \tag{7}$$

$$\text{then max\_val} = \text{magnitude} \tag{8}$$

$$\text{best\_delay} = \tau \tag{9}$$

2.4 Importance of Delay Calculation

Accurate delay calculation ensures that the phase shifts introduced by delays in the measurement setup are properly accounted for. This alignment is crucial for the accurate representation and analysis of the S_{11} parameter, leading to more reliable calibration and fitting results.

3 Basis Function Generation

The function `generate_basis_functions` generates the basis functions used for polynomial fitting. These basis functions form the design matrix for the fitting process. The choice of basis functions depends on the value of `nfit`. Let's go through the function and explain the equations used for each case.

3.1 Function Definition

```
def generate_basis_functions(freqq2, np_data, nfit, f0, dfreq, fcen):
    fitf = np.zeros((np_data * nfit))
    for j in range(nfit):
        for i in range(np_data):
            if nfit > 16:
                if j == 0:
                    fitf[i + j * np_data] = 1
                elif j % 2:
                    fitf[i + j * np_data] = np.cos((freqq2[i] - f0) * 2.0 * np.pi * (j // 2 + 1) / dfr
                else:
                    fitf[i + j * np_data] = np.sin((freqq2[i] - f0) * 2.0 * np.pi * (j // 2) / dfreq)
            else:
                fitf[i + j * np_data] = np.log10(freqq2[i] / fcen) ** j
    return fitf
```

3.2 Explanation

3.2.1 Parameters

- `freqq2`: The frequency values.
- `np_data`: The number of data points.
- `nfit`: The number of basis functions (polynomial terms).
- `f0`: The reference frequency (typically the starting frequency).
- `dfreq`: The frequency range (difference between the maximum and minimum frequency).
- `fcen`: The center frequency (average of the maximum and minimum frequency).

3.2.2 Basis Function Generation

Initialization:

```
fitf = np.zeros((np_data * nfit))
```

This initializes an array `fitf` of zeros with size `np_data` × `nfit` to store the basis functions.

Loop Over Polynomial Terms:

```
for j in range(nfit):

for i in range(np_data):
```

The outer loop iterates over the polynomial terms, and the inner loop iterates over the data points.

Generating Basis Functions for `nfit` > 16:

```
if nfit > 16:

if j == 0:

fitf[i + j * np_data] = 1

elif j % 2:

fitf[i + j * np_data] = np.cos((freqq2[i] - f0) * 2.0 * np.pi * (j // 2 + 1) / dfreq)

else:

fitf[i + j * np_data] = np.sin((freqq2[i] - f0) * 2.0 * np.pi * (j // 2) / dfreq)
```

- For $j = 0$, the basis function is constant:
$$\phi_0(f_i) = 1$$
- For odd j, the basis function is a cosine function:
$$\phi_j(f_i) = \cos\left(\frac{2\pi(j//2+1)(f_i - f_0)}{\Delta f}\right)$$
- For even $j > 0$, the basis function is a sine function:
$$\phi_j(f_i) = \sin\left(\frac{2\pi(j//2)(f_i - f_0)}{\Delta f}\right)$$

Generating Basis Functions for `nfit` ≤ 16:

```
else:

fitf[i + j * np_data] = np.log10(freqq2[i] / fcen) ** j
```

$$\phi_j(f_i) = \left(\log_{10}\left(\frac{f_i}{f_{\text{cen}}}\right)\right)^j$$

3.3 Why Logarithmic Terms?

Logarithmic terms are used for fitting when `nfit` is relatively small (`nfit` ≤ 16) because they can better capture variations in data that span several orders of magnitude. This is particularly useful in scenarios like signal processing where the dynamic range of the data can be very large. The logarithmic terms help in stabilizing the fitting process by compressing the scale of the data.

For `nfit` > 16, trigonometric terms (cosine and sine) are used to capture periodic variations in the data. These basis functions are particularly useful in representing signals that have periodic components.

3.4 Summary

- **Logarithmic Basis Functions**:

$$\phi_j(f_i) = \left(\log_{10}\left(\frac{f_i}{f_{\text{cen}}}\right)\right)^j \quad \text{for } nfit \leq 16$$

- **Trigonometric Basis Functions**:

$$\begin{cases} \phi_0(f_i) = 1 \\ \phi_j(f_i) = \cos\left(\frac{2\pi(j//2+1)(f_i - f_0)}{\Delta f}\right) & \text{for odd } j \\ \phi_j(f_i) = \sin\left(\frac{2\pi(j//2)(f_i - f_0)}{\Delta f}\right) & \text{for even } j > 0 \end{cases} \qquad \text{for } nfit > 16$$

This function effectively generates the appropriate basis functions needed for fitting the data, considering the characteristics of the data and the desired complexity of the fit.

4 Polynomial Fitting Using Weighted Least Squares and QR Decomposition

The `polyfitr` function fits a polynomial to the given data using a weighted least squares approach and QR decomposition to solve the resulting normal equations. Below is a detailed explanation of the function.

4.1 Initialization of Design Matrix

The design matrix $\mathbf{X}$ is constructed using the basis functions. This matrix has dimensions $n \times p$, where n is the number of data points (`nfreq`) and p is the number of polynomial terms (`npoly`).

```
for i in range(nfreq):
    for j in range(npoly):
        mcalc[i * npoly + j] = fitfun(j, i, nfreq, fitfn)
```

Mathematically, the design matrix $\mathbf{X}$ is:

$$\mathbf{X}_{ij} = \phi_j(f_i)$$

where ϕ_j is the j-th basis function evaluated at the i-th data point f_i.

4.2 Construction of Normal Equations

The normal equations are constructed using the design matrix $\mathbf{X}$ and the data vector $\mathbf{y}$ (which is `ddata` in the code). The weighted normal equations are:

$$\mathbf{X}^T\mathbf{W}\mathbf{X}\mathbf{a} = \mathbf{X}^T\mathbf{W}\mathbf{y}$$

where $\mathbf{W}$ is the diagonal weight matrix with `wtt[k]` as its diagonal elements.

```
for i in range(npoly):
    re = 0.0
    for k in range(nfreq):
        if wtt[k] > 0:
            re += mcalc[i + k * npoly] * ddata[k] * wtt[k]
    bbrr[i] = re

    for j in range(npoly):
        re = 0.0
        for k in range(nfreq):
            if wtt[k] > 0:
                re += mcalc[i + k * npoly] * mcalc[j + k * npoly] * wtt[k]
        aarr[i * npoly + j] = re
```

Here: - $\mathbf{A} = \mathbf{X}^T\mathbf{W}\mathbf{X}$ is stored in `aarr`. - $\mathbf{b} = \mathbf{X}^T\mathbf{W}\mathbf{y}$ is stored in `bbrr`.

4.3 Solving Normal Equations Using QR Decomposition

The normal equations $\mathbf{Aa} = \mathbf{b}$ are solved using QR decomposition. QR decomposition decomposes $\mathbf{A}$ into an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$:

$$\mathbf{A} = \mathbf{QR}$$

The normal equations then become:

$$\mathbf{Ra} = \mathbf{Q}^T\mathbf{b}$$

This system is solved for the coefficient vector $\mathbf{a}$.

```
qrd(aarr, npoly, bbrr)
```

4.4 Calculation of Fitted Values

The fitted values $\hat{y}_i$ are calculated using the coefficient vector $\mathbf{a}$ and the basis functions. This gives us the polynomial fit.

```
for i in range(nfreq):
    re = 0.0
    for j in range(npoly):
        re += bbrr[j] * fitfun(j, i, nfreq, fitfn)
    dataout[i] = re
```

Mathematically:

$$\hat{y}_i = \sum_{j=0}^{p-1} a_j \phi_j(f_i)$$

4.5 Summary of Key Equations

- **Design Matrix:**
$$\mathbf{X}_{ij} = \phi_j(f_i)$$
- **Normal Equations:**
$$\mathbf{X}^T\mathbf{W}\mathbf{X}\mathbf{a} = \mathbf{X}^T\mathbf{W}\mathbf{y}$$
- **QR Decomposition:**
$$\mathbf{A} = \mathbf{QR}$$
- **Solving for Coefficients:**
$$\mathbf{Ra} = \mathbf{Q}^T\mathbf{b}$$
- **Fitted Values:**
$$\hat{y}_i = \sum_{j=0}^{p-1} a_j \phi_j(f_i)$$

4.6 Example

Suppose we have 3 data points (f_1, y_1), (f_2, y_2), (f_3, y_3) and we want to fit a polynomial of degree 2. The basis functions might be $\phi_0(f) = 1$, $\phi_1(f) = \log_{10}(f/f_{\text{cen}})$, and $\phi_2(f) = \log_{10}(f/f_{\text{cen}})^2$.

The design matrix $\mathbf{X}$ would be:

$$\mathbf{X} = \begin{bmatrix} 1 & \log_{10}(f_1/f_{\text{cen}}) & \log_{10}(f_1/f_{\text{cen}})^2 \\ 1 & \log_{10}(f_2/f_{\text{cen}}) & \log_{10}(f_2/f_{\text{cen}})^2 \\ 1 & \log_{10}(f_3/f_{\text{cen}}) & \log_{10}(f_3/f_{\text{cen}})^2 \end{bmatrix}$$

The normal equations are then solved to find the coefficients a_0, a_1, a_2, which are used to calculate the fitted values $\hat{y}_1, \hat{y}_2, \hat{y}_3$.

This function `polyfitr` encapsulates this process, allowing for efficient polynomial fitting of complex-valued data.

5 QR Decomposition Function (`qrd`)

The `qrd` function performs QR decomposition to solve the normal equations. Here's how it works:

1. **Initialization:** The function starts by converting the input array `aarr` into a matrix $\mathbf{A}$ and the array `bbrr` into a vector $\mathbf{b}$:
$$\mathbf{A} = \text{np.array(aarr, dtype=np.float64).reshape((npoly, npoly))}$$
$$\mathbf{b} = \text{np.array(bbrr, dtype=np.float64)}$$
2. **QR Decomposition:** QR decomposition decomposes matrix $\mathbf{A}$ into an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$:
$$\mathbf{A} = \mathbf{QR}$$
This is performed using:
$$\mathbf{Q}, \mathbf{R} = \text{np.linalg.qr(A)}$$
3. **Solving the System:** The normal equations $\mathbf{Aa} = \mathbf{b}$ become:
$$\mathbf{QRa} = \mathbf{b}$$
Multiplying both sides by $\mathbf{Q}^T$ (transpose of $\mathbf{Q}$):
$$\mathbf{Ra} = \mathbf{Q}^T\mathbf{b}$$
This system is upper triangular and can be solved using back-substitution:
$$\mathbf{x} = \text{np.linalg.solve(R, Q.T @ b)}$$
4. **Updating the Solution Vector:** The solution vector $\mathbf{x}$ contains the coefficients $\mathbf{a}$ of the polynomial. These coefficients are then copied back into the `bbrr` array:
$$\text{for i in range(len(bbrr)):}$$
$$\text{bbrr[i] = x[i]}$$

Detailed Steps with Equations

1. **Matrix Initialization:**
$$\mathbf{A} \in \mathbb{R}^{p\times p}, \quad \mathbf{b} \in \mathbb{R}^p$$
Here, p is the number of polynomial terms (`npoly`).
2. **QR Decomposition:** QR decomposition factorizes $\mathbf{A}$ as:
$$\mathbf{A} = \mathbf{QR}$$
where:
   - $\mathbf{Q} \in \mathbb{R}^{p\times p}$ is an orthogonal matrix ($\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$).
   - $\mathbf{R} \in \mathbb{R}^{p\times p}$ is an upper triangular matrix.
3. **Solving the System:** The transformed system is:
$$\mathbf{Ra} = \mathbf{Q}^T\mathbf{b}$$
Solving this involves back-substitution since $\mathbf{R}$ is upper triangular.
4. **Back-Substitution:** For an upper triangular matrix $\mathbf{R}$, solving $\mathbf{Ra} = \mathbf{c}$ can be done step-by-step starting from the last equation:
$$\begin{aligned} r_{pp}a_p &= c_p \\ r_{(p-1)p}a_p + r_{(p-1)(p-1)}a_{(p-1)} &= c_{(p-1)} \\ &\vdots \\ r_{1p}a_p + r_{1(p-1)}a_{(p-1)} + \cdots + r_{11}a_1 &= c_1 \end{aligned}$$
This allows solving for each a_i sequentially.

Example

Suppose $\mathbf{A}$ is a 3x3 matrix and $\mathbf{b}$ is a 3x1 vector:

$$\mathbf{A} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Perform QR decomposition:

$$\mathbf{A} = \mathbf{QR}$$

where $\mathbf{Q}$ and $\mathbf{R}$ are:

$$\mathbf{Q} = \begin{bmatrix} q_{11} & q_{12} & q_{13} \\ q_{21} & q_{22} & q_{23} \\ q_{31} & q_{32} & q_{33} \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{bmatrix}$$

Solve $\mathbf{Ra} = \mathbf{Q}^T\mathbf{b}$ using back-substitution to find $\mathbf{a}$.

This is the core process behind the `qrd` function, allowing it to solve for polynomial coefficients efficiently.

6 Conclusion

The presented document outlines the calibration process of T_{sky} using polynomial fitting and QR decomposition. The theory and implementation details are crucial for accurate and reliable data analysis in radio astronomy.