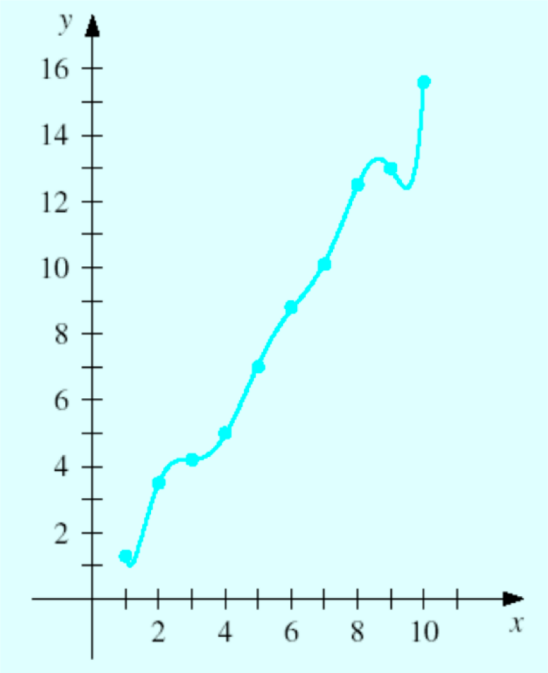
NA-Data Fitting Practice

Data Fitting

- **Approximate fit**
  - Design a model that generalizes the patterns of the input data



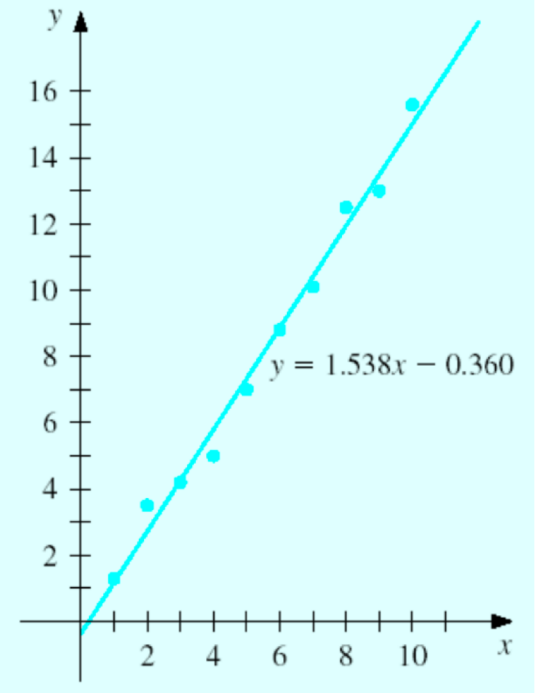


Image Transformation Method

- Translation
- Rotation
- Scale
- Similarity
- Affine
- Perspective(Projective)

Image Transformation Method

Translation

Rotation

Scale

Simiarlity

Affine

Homography

Image Transformation Method

- **Translation**

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x + dx \\ y + dy \end{pmatrix}$$

- **Rotation**

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{pmatrix}$$

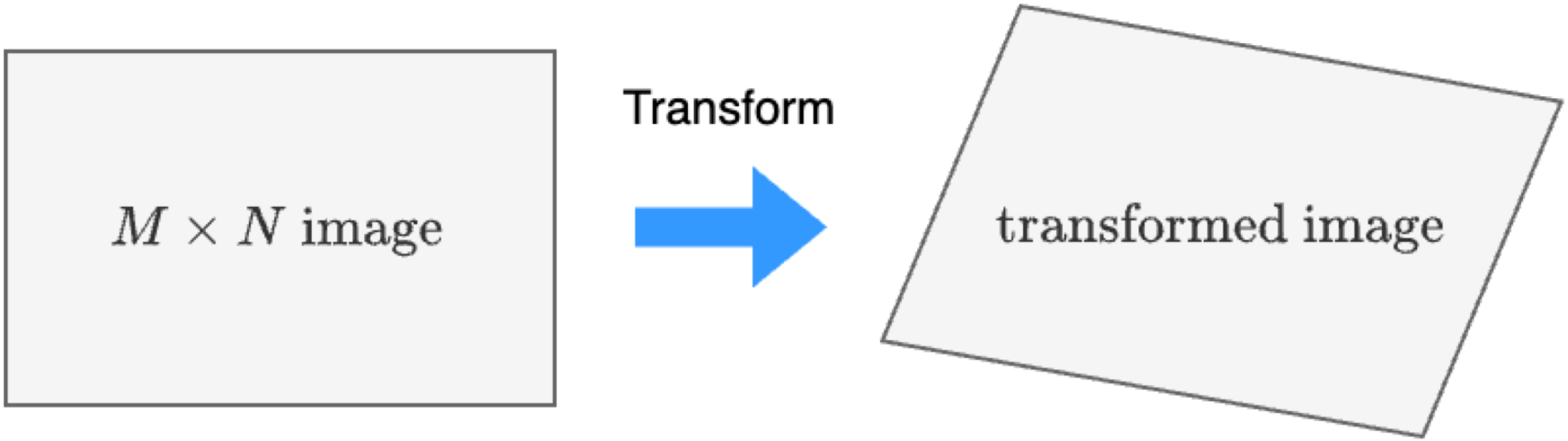
- **Scale**

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx \\ sy \end{pmatrix}$$

Similarity Transform in 2D Image

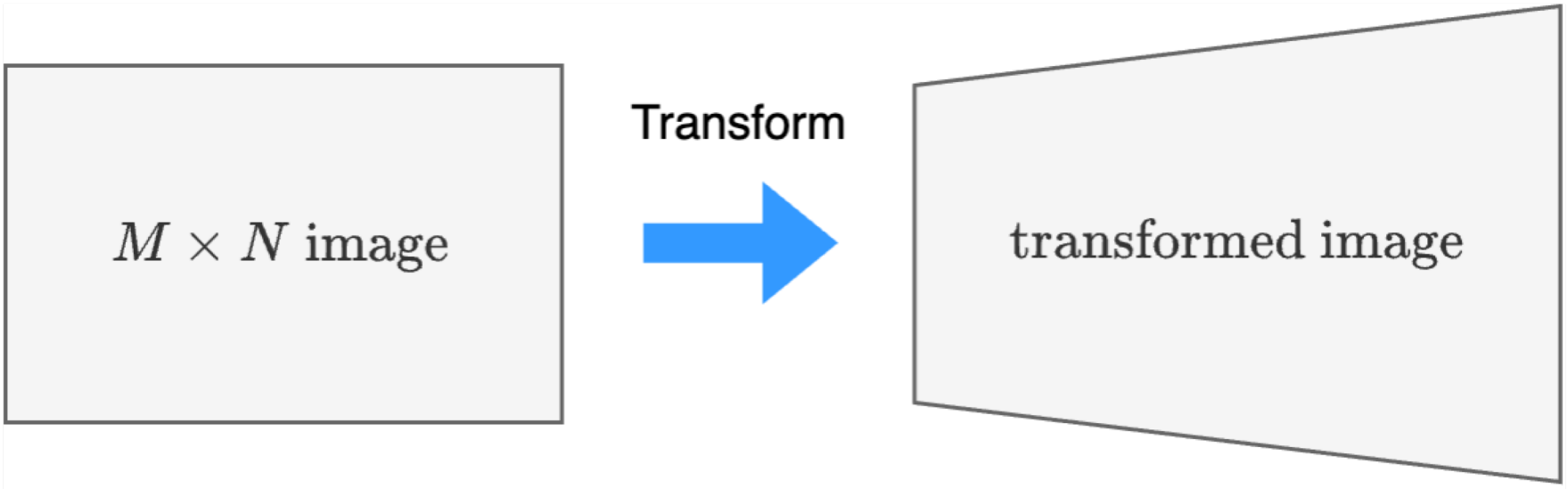
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} sx\cos\theta - sy\sin\theta + dx \\ sx\sin\theta + sy\cos\theta + dy \end{pmatrix}$$

Affine Transform in 2D Image



$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a_0 x + a_1 y + a_2 \\ a_3 x + a_4 y + a_5 \end{pmatrix}$$

Perspective Transform in 2D Image



$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{h_6 x + h_7 y + 1} \begin{pmatrix} h_0 x + h_1 y + h_2 \\ h_3 x + h_4 y + h_5 \end{pmatrix}$$

→This transform is called **homography**

How to compute Homography?

- **Transformed Position**

$$x' = \frac{h_0 x + h_1 y + h_2}{h_6 x + h_7 y + 1}$$

$$y' = \frac{h_3 x + h_4 y + h_5}{h_6 x + h_7 y + 1}$$

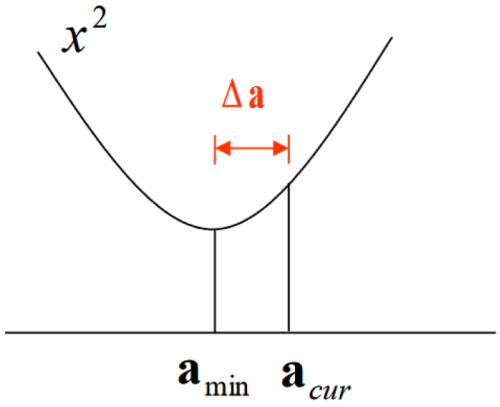
- **Need to determine 8 parameters**

$$\rightarrow [h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8]$$

→ Solution is Non-linear Least-Square fitting using LM method

Revisiting Levenberg-Marquardt method

$\mathbf{a}_{cur}$: Near the minimum



$$\mathbf{a}_{\min} = \mathbf{a}_{cur} + \mathbf{H}^{-1}[-\nabla x^2(\mathbf{a}_{cur})]$$

$$\Delta \mathbf{a} = \mathbf{a}_{\min} - \mathbf{a}_{cur} = \mathbf{H}^{-1}[-\nabla x^2(\mathbf{a}_{cur})]$$

$$\therefore \underline{\mathbf{H}}\ \underline{\Delta \mathbf{a}} = -\underline{\nabla x^2}$$

Hessian matrix — update term — gradient

Inverse-Hessian method

$\mathbf{a}_{cur}$: Far from the minimum

$$\Delta \mathbf{a} = -const \times \nabla x^2(\mathbf{a}_{cur})$$

Steepest Descent method

Revisiting Levenberg-Marquardt method

- Algorithm

i) Guess $\mathbf{a}_{cur}$

ii) compute $x^2(\mathbf{a}_{cur})$

iii) pick a modest λ, say $\lambda = 0.001$

iv) solve

$$H' \cdot \Delta a = -\nabla x^2(a_{cur}) \qquad \text{①}$$

where $H' = H + \lambda I$

v) if $\mathbf{x}^2(\mathbf{a}_{cur} + \Delta\mathbf{a}) \geq \mathbf{x}^2(\mathbf{a}_{cur})$

increase λ by a factor of 10 and go to iv)

else

decrease λ by a factor of 10 and

update $\mathbf{a}_{cur} = \mathbf{a}_{cur} + \Delta\mathbf{a}$ and go to iv)

How to compute Homography?

How to compute Homography?

```python
import numpy as np
import cv2

def compute_residuals(params, pts1, pts2):
    residuals = []
    for p1, p2 in zip(pts1, pts2):
        x1, y1 = p1
        x2, y2 = p2
        norm = params[6] * x1 + params[7] * y1 + 1
        x1_proj = (params[0] * x1 + params[1] * y1 + params[2]) / norm
        y1_proj = (params[3] * x1 + params[4] * y1 + params[5]) / norm
        residuals.append((x2 - x1_proj)**2)
        residuals.append((y2 - y1_proj)**2)
    return np.array(residuals)
```

How to compute Homography?

```python
def compute_jacobian(params, pts1, pts2):
    J = []
    for p1, p2 in zip(pts1, pts2):
        x1, y1 = p1
        x2, y2 = p2
        norm = params[6] * x1 + params[7] * y1 + 1
        x1_proj = (params[0] * x1 + params[1] * y1 + params[2]) / norm
        y1_proj = (params[3] * x1 + params[4] * y1 + params[5]) / norm

        d_x_proj = [
            2 * (x1_proj - x2) * x1 / norm,  # d/d(h0)
            2 * (x1_proj - x2) * y1 / norm,  # d/d(h1)
            2 * (x1_proj - x2) * 1 / norm,   # d/d(h2)
            0, 0, 0,  # Terms independent of h3, h4, h5
            -2 * (x1_proj - x2) * x1_proj * x1 / norm,  # d/d(h6)
            -2 * (x1_proj - x2) * x1_proj * y1 / norm   # d/d(h7)
        ]
        d_y_proj = [
            0, 0, 0,  # Terms independent of h0, h1, h2
            2 * (y1_proj - y2) * x1 / norm,  # d/d(h3)
            2 * (y1_proj - y2) * y1 / norm,  # d/d(h4)
            2 * (y1_proj - y2) * 1 / norm,   # d/d(h5)
            -2 * (y1_proj - y2) * y1_proj * x1 / norm,  # d/d(h6)
            -2 * (y1_proj - y2) * y1_proj * y1 / norm   # d/d(h7)
        ]
        J.append(d_x_proj)
        J.append(d_y_proj)
    return np.array(J)
```

How to compute Homography?

```python
def compute_homography(pts1, pts2):
    params = np.array([1, 0, 0, 0, 1, 0, 0, 0], dtype=np.float64)

    max_iter = 100
    lambda_factor = 1e-3
    tolerance = 1e-6

    for iteration in range(max_iter):
        residuals = compute_residuals(params, pts1, pts2)
        J = compute_jacobian(params, pts1, pts2)

        H = J.T @ J
        H_lm = H + lambda_factor * np.eye(len(H))

        g = J.T @ residuals
        delta = np.linalg.solve(H_lm, -g)

        params_new = params + delta

        if np.linalg.norm(delta) < tolerance:
            print(f"Converged after {iteration} iterations.")
            break

        residuals_new = compute_residuals(params_new, pts1, pts2)
        if np.linalg.norm(residuals_new) < np.linalg.norm(residuals):
            lambda_factor /= 10
            params = params_new
        else:
            lambda_factor *= 10

    H_optimized = np.array([
    [params[0], params[1], params[2]],
    [params[3], params[4], params[5]],
    [params[6], params[7], 1]
    ])
    return H_optimized
```

- Algorithm

i) Guess $\mathbf{a}_{cur}$

ii) compute $x^2(\mathbf{a}_{cur})$

iii) pick a modest λ, say $\lambda = 0.001$

iv) solve

$$H' \cdot \Delta a = -\nabla x^2(a_{cur}) \quad \text{——— ①}$$

where $H' = H + \lambda I$

v) if $\mathbf{x}^2(\mathbf{a}_{cur} + \Delta\mathbf{a}) \geq \mathbf{x}^2(\mathbf{a}_{cur})$

increase λ by a factor of 10 and go to iv)

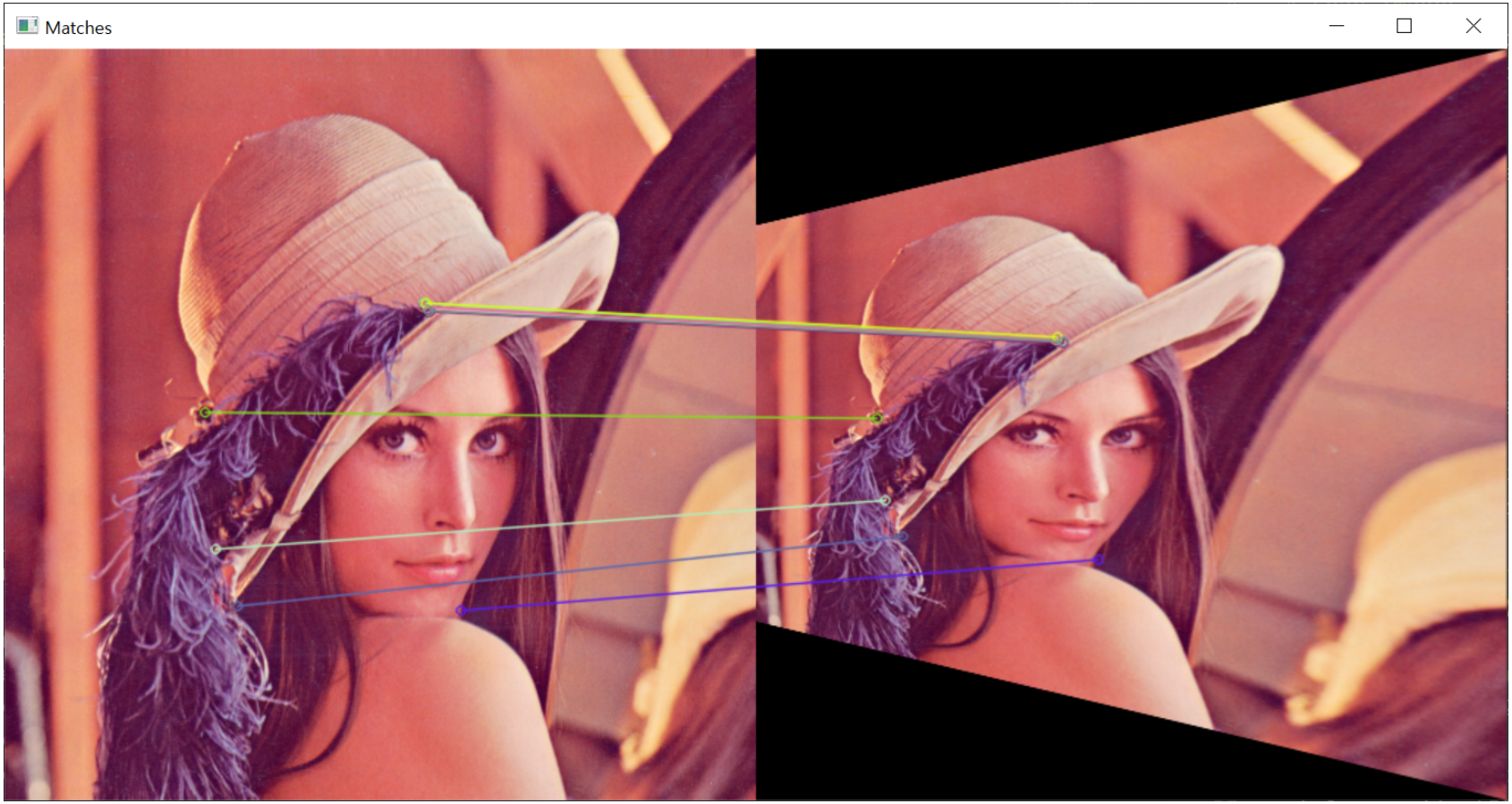
else

decrease λ by a factor of 10 and

update $\mathbf{a}_{cur} = \mathbf{a}_{cur} + \Delta\mathbf{a}$ and go to iv)

How to compute Homography?

- **Find correspondence in Two Images**

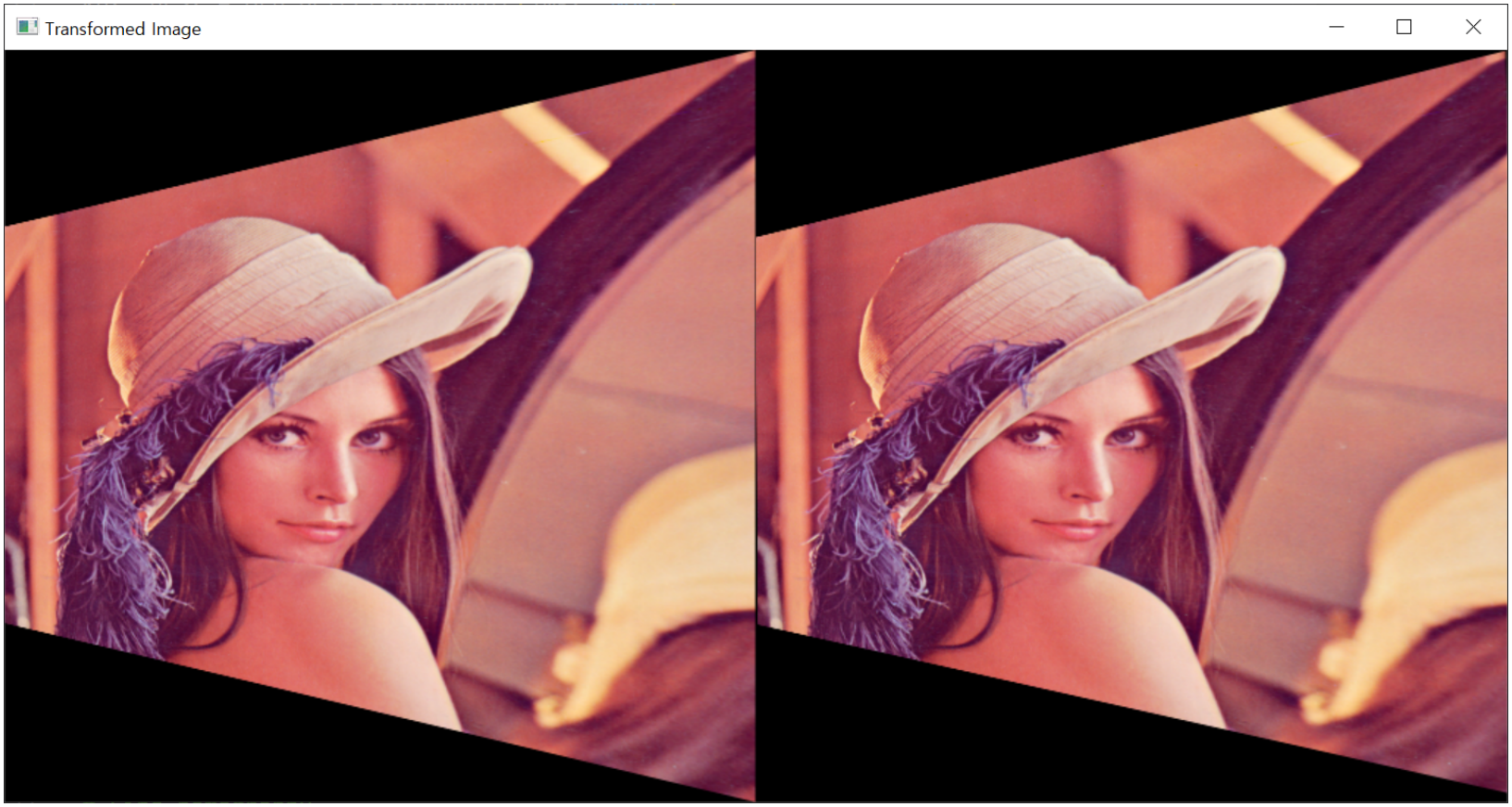


How to compute Homography?

```python
def main():
    img1 = cv2.imread('target_img.png', cv2.IMREAD_GRAYSCALE)
    img2 = cv2.imread('transformed_img.png', cv2.IMREAD_GRAYSCALE)

    orb = cv2.ORB_create()
    kp1, des1 = orb.detectAndCompute(img1, None)
    kp2, des2 = orb.detectAndCompute(img2, None)

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)

    pts1 = np.float32([kp1[m.queryIdx].pt for m in matches[:10]])
    pts2 = np.float32([kp2[m.trainIdx].pt for m in matches[:10]])

    H_optimized = compute_homography(pts1, pts2)
    print(H_optimized)

    transformed_image = cv2.warpPerspective(img1, H_optimized, (img1.shape[1], img2.shape[0]))
    cv2.imshow('Transformed Image', transformed_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

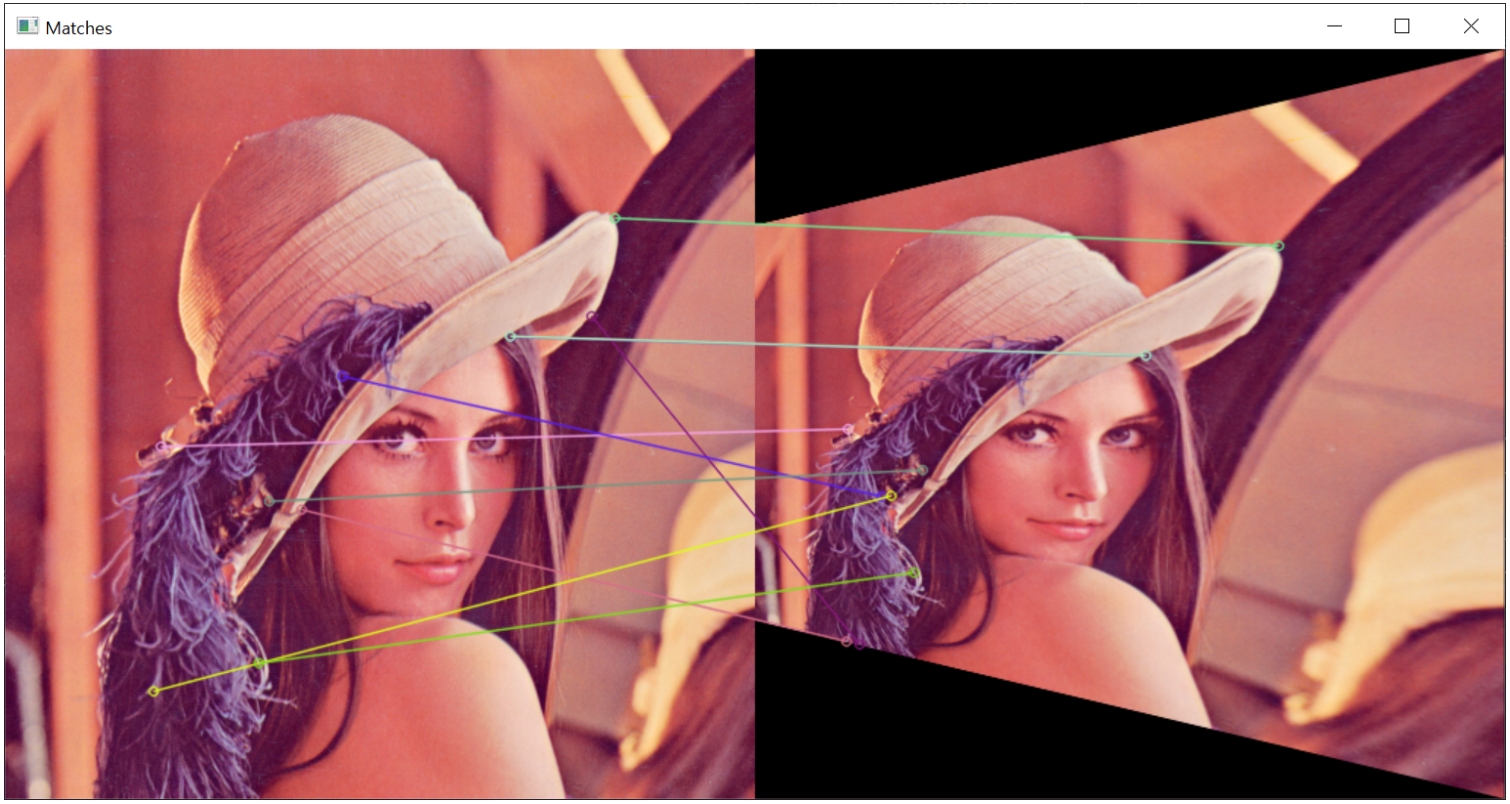
How to compute Homography?

- **Result**



How to compute Homography?

- **Worst case**

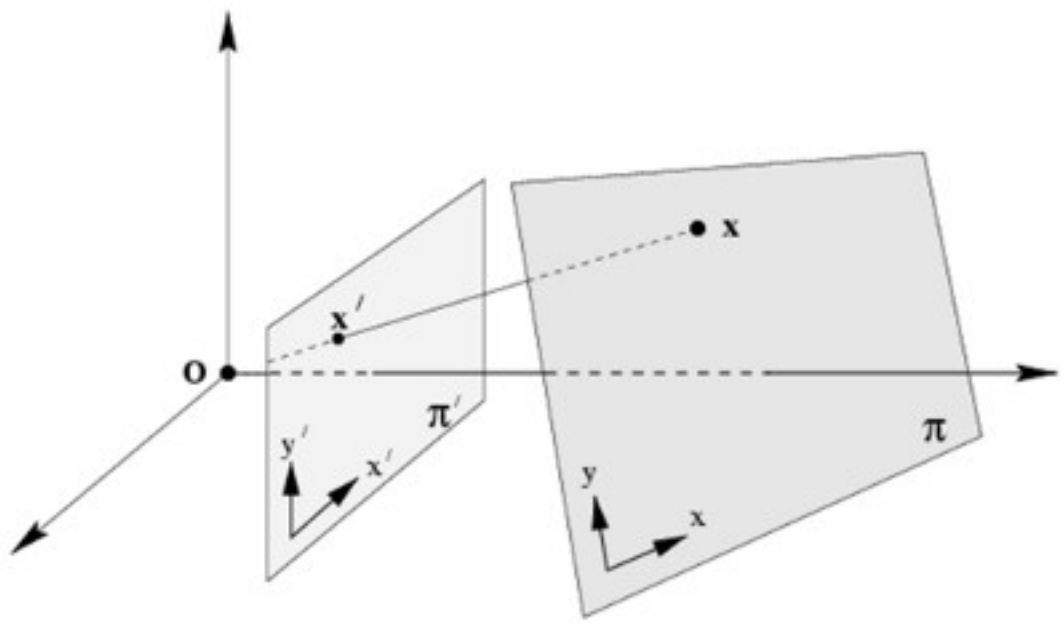


→ **Solution is RANSAC**

How to compute homography: Other Solution

- **General Form in Homogenous Coordinate**

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$



How to compute homography: Other Solution

- **General Form in Homogenous Coordinate**

$$\underbrace{\begin{bmatrix} x^{(1)} & y^{(1)} & 1 & 0 & 0 & 0 & -\hat{x}^{(1)}x^{(1)} & -\hat{x}^{(1)}y^{(1)} \\ 0 & 0 & 0 & x^{(1)} & y^{(1)} & 1 & -\hat{y}^{(1)}x^{(1)} & -\hat{y}^{(1)}y^{(1)} \\ \cdots & & & & & & & \\ x^{(i)} & y^{(i)} & 1 & 0 & 0 & 0 & -\hat{x}^{(i)}x^{(i)} & -\hat{x}^{(i)}y^{(i)} \\ 0 & 0 & 0 & x^{(i)} & y^{(i)} & 1 & -\hat{y}^{(i)}x^{(i)} & -\hat{y}^{(i)}y^{(i)} \\ \cdots & & & & & & & \\ x^{(n)} & y^{(n)} & 1 & 0 & 0 & 0 & -\hat{x}^{(n)}x^{(n)} & -\hat{x}^{(n)}y^{(n)} \\ 0 & 0 & 0 & x^{(n)} & y^{(n)} & 1 & -\hat{y}^{(n)}x^{(n)} & -\hat{y}^{(n)}y^{(n)} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}}_{h} = \underbrace{\begin{bmatrix} \hat{x}^{(1)} \\ \hat{y}^{(1)} \\ \cdots \\ \hat{x}^{(i)} \\ \hat{y}^{(i)} \\ \cdots \\ \hat{x}^{(n)} \\ \hat{y}^{(n)} \end{bmatrix}}_{b}$$