```python
import numpy as np
```

Implementation

```python
def householder_reflection(a):
    """
    Create a Householder reflection matrix for a vector a.

    Parameters:
    a (numpy.ndarray): Input vector.

    Returns:
    v (numpy.ndarray): Householder reflection vector.
    """

    v = a.copy()
    v[0] += np.sign(a[0]) * np.linalg.norm(a)
    v = v / np.linalg.norm(v)
    return v


def golub_kahan_bidiagonalization(A):

    """
    Perform Golub-Kahan bidiagonalization on matrix A.

    Parameters:
    A (numpy.ndarray): Input matrix with shape (m, n).

    Returns:
    B (numpy.ndarray): Bidiagonal matrix.
    U (numpy.ndarray): Orthogonal matrix U.
    V (numpy.ndarray): Orthogonal matrix V.
    """

    m, n = A.shape
    U = np.eye(m)
    V = np.eye(n)
    B = A.copy()

    for i in range(min(m, n)):
        # Hosholder's reflection from the left
        x = B[i:, i]
        v = householder_reflection(x)
        H = np.eye(m - i) - 2 * np.outer(v, v)
        B[i:, :] = np.dot(H, B[i:, :])
        U[:, i:] = np.dot(U[:, i:], H)
        if i < n - 1:
            # Hosholder's reflection from the right
            x = B[i, i+1:]
            v = householder_reflection(x)
            H = np.eye(n - (i+1)) - 2 * np.outer(v, v)
            B[:, i+1:] = np.dot(B[:, i+1:], H)
            V[:, i+1:] = np.dot(V[:, i+1:], H)



    return B, U, V

def compute_svd_from_bidiagonal(B, U, V):
    m, n = B.shape
    U_b, Sigma, Vt_b = np.linalg.svd(B)
    U = np.dot(U, U_b)
    V = np.dot(V, Vt_b.T)
    return U, Sigma, V.T



def reconstruct_matrix(U, Sigma, Vt):
    Sigma_mat = np.zeros((U.shape[0], Vt.shape[1]))
    np.fill_diagonal(Sigma_mat, Sigma)
    A_reconstructed = np.dot(U, np.dot(Sigma_mat, Vt))
    return A_reconstructed
```

Example

```
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=float)
A
```

```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

```
B, U, V = golub_kahan_bidiagonalization(A)
U_svd, Sigma, Vt_svd = compute_svd_from_bidiagonal(B, U, V)
A_reconstructed = reconstruct_matrix(U_svd, Sigma, Vt_svd)
A_reconstructed
```

```
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
```

```
np.round(Sigma, 3)
```

```
array([16.848,  1.068,  0.   ])
```