In [ ]:
```
import numpy as np
```

In [ ]:
```
# Define a 100 x 15 design matrix
m = 100
n = 15
t = np.linspace(0, 1, 100)
A = np.vander(t, n)
```

In [ ]:
```
# An algorithm to compute the psuedoinverse including only columns whose associated
# singular values exceed the tolerance `tol` as defined in the algorithm below to
# account for stability

# Compute the SVD of A
U, S, Vh = np.linalg.svd(A)
# Define a tolerance that is the (largest dim of A) x (largest singular value of
# A or the condition number of the matrix) x (machine epsilon) to account for stabi
tol = np.max(A.shape) * S[0] * np.finfo(float).eps
 # Set r to the count of singular values that exceed the tolerance `tol`
r = np.sum(S > tol)
 # Compute S^{-1} including only those values that exceed the tolerance `tol`
S_inv = np.ones(r)/S
# Compute A^+ = VS^{-1}U^*
X = Vh.conj().T @ np.diag(S_inv) @ U[:, :r].conj().T
```

In [ ]:
```
X
```

Out[ ]:
```
array([[ 3.70831427e+06, -4.15780655e+06, -3.67620796e+06, ...,
        -3.67620811e+06, -4.15780575e+06,  3.70831390e+06],
       [-2.69889331e+07,  2.99687959e+07,  2.67522901e+07, ...,
         2.47146224e+07,  2.82404902e+07, -2.49274641e+07],
       [ 8.81785521e+07, -9.68174815e+07, -8.73737028e+07, ...,
        -7.41288593e+07, -8.55835132e+07,  7.47790128e+07],
       ...,
       [ 3.32700651e+03, -1.90770346e+03, -2.64980310e+03, ...,
        -4.55034207e+02, -5.85070087e+02,  4.66179955e+02],
       [-9.37444311e+01,  2.83204146e+01,  5.58349172e+01, ...,
         6.98495601e+00,  9.09169176e+00, -7.17822583e+00],
       [ 8.96680779e-01,  2.34816491e-01, -3.35452898e-02, ...,
        -1.51472783e-02, -2.01271777e-02,  1.56544827e-02]])
```

In [ ]: