```python
import numpy as np
import numpy.linalg as la

attr = np.array([get_movie_attr(j) for j in range(k)])
rates = np.array([get_ratings(i) for i in range(n)]).T
prefs = la.solve(attr, rates)
```

```python
A = np.array([[5, 6],[25,42]])

def lu_decomp(A):
    """(L, U) = lu_decomp(A) is the LU decomposition A = L U
       A is any matrix
       L will be a lower-triangular matrix with 1 on the diagonal, the same shape
       U will be an upper-triangular matrix, the same shape as A
    """
    n = A.shape[0]
    if n == 1:
        L = np.array([[1]])
        U = A.copy()
        return (L, U)

    A11 = A[0,0]
    A12 = A[0,1:]
    A21 = A[1:,0]
    A22 = A[1:,1:]

    L11 = 1
    U11 = A11

    L12 = np.zeros(n-1)
    U12 = A12.copy()

    L21 = A21.copy() / U11
    U21 = np.zeros(n-1)

    S22 = A22 - np.outer(L21, U12)
    (L22, U22) = lu_decomp(S22)

    L = np.block([[L11, L12], [L21, L22]])
    U = np.block([[U11, U12], [U21, U22]])
    return (L, U)
lu_decomp(A)
```

```
(array([[1., 0.],
        [5., 1.]]),
 array([[ 5.,  6.],
        [ 0., 12.]]))
```

```python
def forward_sub(L, b):
    """x = forward_sub(L, b) is the solution to L x = b
       L must be a lower-triangular matrix
       b must be a vector of the same leading dimension as L
    """
    n = L.shape[0]
    x = np.zeros(n)
    for i in range(n):
        tmp = b[i]
        for j in range(i):
            tmp -= L[i,j] * x[j]
        x[i] = tmp / L[i,i]
    return x

def back_sub(U, b):
    """x = back_sub(U, b) is the solution to U x = b
       U must be an upper-triangular matrix
       b must be a vector of the same leading dimension as U
    """
    n = U.shape[0]
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        tmp = b[i]
        for j in range(i+1, n):
            tmp -= U[i,j] * x[j]
        x[i] = tmp / U[i,i]
    return x

def lu_solve(L, U, b):
    """x = lu_solve(L, U, b) is the solution to L U x = b
       L must be a lower-triangular matrix
       U must be an upper-triangular matrix of the same size as L
       b must be a vector of the same leading dimension as L
    """
    y = forward_sub(L, b)
    x = back_sub(U, y)
    return x
```

```python
import numpy as np
ratings = []
for i in range(k):
    r = 0
    for j in range(n):
        r += np.dot(get_friend_prefs(j),get_movie_attr(i))/m
    ratings.append(r)
top = ratings.index(max(ratings))
ratings = np.array(ratings)
```

```python
import numpy as np
import numpy.linalg as la

N = ion_locations.shape[0]

A = np.zeros((N,N))
for i in range(N):
    for j in range(N):
        A[i][j] = (test_locations[i][2]-ion_locations[j][2])/(la.norm(test_locations[i]-ion_locations[j],2)**3
            )

charges = la.solve(A,E_field)
```


```python
import numpy as np
import numpy.linalg as la

N = ion_locations.shape[0]

A = np.zeros((N,N))
for i in range(N):
    for j in range(N):
        A[i][j] = (test_locations[i][2]-ion_locations[j][2])/(la.norm(test_locations[i]-ion_locations[j],2)**3
            )

charges = la.solve(A,E_field)
```


```python
import numpy as np
n = 100000
V = 6 * 6 * 2
x = 6 * np.random.rand(n) - 3
y = 6 * np.random.rand(n) - 3
z = 2 * np.random.rand(n)

def check_in(x, y, z):
    if abs(x) + abs(y) < 3 and z > 0 and z < f(x,y):
        return 1
    return 0

n_in = np.sum(np.array([check_in(x[i], y[i], z[i]) for i in range(n)]))
volume = V * n_in/n
```