```
r"""
Updates to the thin SVD using NumPy.

This function is a SAGE replication of Matthew Brand's article on "Fast low-rank modifications
 of the thin singular value decomposition." <http://www.stat.osu.edu/~dmsl/thinSVDtracking.pdf>
This function is an approximation to the true thin SVD, therefore, no tests are provided.

AUTHORS:

- Taylor Steiger, James Pak (2013-06-10): initial version


EXAMPLES::

    Update

    sage: X = np.array([[1.0,2.0,3.0,4.0],[3.0,2.0,5.0,5.0],[5.0,3.0,1.0,1.0],[7.0,7.0,7.0,7.0]])
    sage: U, s, V = np.linalg.svd(X, full_matrices = False)
    sage: a = np.reshape(np.array([4.0,5.0,1.0,7.0]), (-1, 1))
    sage: U, S, V = svd_update(U, np.diag(s), V, X, a, update = True)

    Downdate

    sage: X = np.array([[1.0,2.0,3.0,4.0],[3.0,2.0,5.0,5.0],[5.0,3.0,1.0,1.0],[7.0,7.0,7.0,7.0]])
    sage: U, s, V = np.linalg.svd(X, full_matrices = False)
    sage: U, S, V = svd_update(U, np.diag(s), V, X, downdate = True)

    Revise

    sage: X = np.array([[1.0,2.0,3.0,4.0],[3.0,2.0,5.0,5.0],[5.0,3.0,1.0,1.0],[7.0,7.0,7.0,7.0]])
    sage: U, s, V = np.linalg.svd(X, full_matrices = False)
    sage: a = np.reshape(np.array([4.0,5.0,1.0,7.0]), (-1, 1))
    sage: U, S, V = svd_update(U, np.diag(s), V, X, a)

    Recenter

    sage: X = np.array([[1.0,2.0,3.0,4.0],[3.0,2.0,5.0,5.0],[5.0,3.0,1.0,1.0],[7.0,7.0,7.0,7.0]])
    sage: U, s, V = np.linalg.svd(X, full_matrices = False)
    sage: U, S, V = svd_update(U, np.diag(s), V, X)

"""

#*****************************************************************************
#       Copyright (C) 2013 Taylor Steiger <tsteiger@uw.edu>
#       Copyright (C) 2013 James Pak <jimmypak@uw.edu>
#
#  Distributed under the terms of the GNU General Public License (GPL)
#  as published by the Free Software Foundation; either version 2 of
#  the License, or (at your option) any later version.
#                  http://www.gnu.org/licenses/
#*****************************************************************************

import numpy as np

def svd_update(U, S, V, X, c = None, update = False, downdate = False):
    """
    INPUT:

    - U -- a (nxn) matrix containing singular vectors of X.

    - S -- a (nxn) diagonal matrix containing singular values. the ith diagonal entry is
      the singular value corresponding to the ith column of U.

    - V -- a (nxn) matrix containing singular vectors of X.

    - X -- a (mxn or nxm) matrix such that U^T*X*V=S.

    - c -- (default: None) a column vector for revision or update of decomposition.

    - update -- (default: False) boolean whether to add c to the decomposition. If true,
      c must also be provided.

    - downdate -- (default: False) boolean whether to downdate the decomposition.

    OUTPUT:

    A 3-tuple consisting of matrices in this order:

    1. Transformed U.
    2. Transformed S.
    3. Transformed V.

    ALGORITHM:

    The SVD rank-1 modification algorithm is described by Matthew Brand
    in the paper at, <http://www.stat.osu.edu/~dmsl/thinSVDtracking.pdf>.
    The algorithm works as follows:

    #. Extend V so that both its last column and row are the zero vector.
    #. Compute a and b so that they perform the appropriate transformation.
    #. If updating:
    #.    a = c, b^T = [0,...,0,1]
    #. Else if downdating:
    #.    a = X[:,-1], b^T = [0,...,0,1]
    #. Else if revising:
    #.    a = X[:,-1] - c, b^T = [0,...,0,1]
    #. Else: #recentering
    #.    a = X * (I - (1/q) * (l * l^T)) # l = [1,...,1]
    #. Compute m, p, Ra and P.
    #. m = U^T * a
    #. p = a - U * m
    #. Ra = ||p||
    #. P = Ra^(-1) * p
    #. Compute n, q, Rb and Q, similarly to the previous step with substitution of b for a.
    #. Compute K.
    #. K = [S 0] * [m ] *[n ]^T
    #.     [0 0]   [Ra]  [Rb]
    #. Diagonalize K.

    WARNING::

    During testing, this code showed strong deviations, since it is an approximation, from
    the true thin SVD of the matrix X. However, testing was conducted with a small matrix X,
    and may be working perfectly fine.
    """

    V = np.vstack([V, np.zeros(V.shape[1])])
    if down or type(c) == type(np.array([])):
        b = np.zeros(V.shape[0])
        b[-1] = 1
        b = np.reshape(b, (b.shape[0], 1))
        if down:
            a = np.reshape(np.multiply(X[:,-1], -1), (-1, 1))
        elif add:
            a = np.reshape(c, (-1, 1))
        else:
            a = np.reshape(X[:,-1] - c, (-1, 1))
    else:
        ones = np.zeros(V.shape[0])
        ones = np.add(b, 1)
        b = np.reshape(ones, (-1, 1))
        a = np.reshape(np.multiply((-1/X.shape[1]), np.dot(X, b)), (-1, 1))

    m = np.reshape(np.dot(np.transpose(U), a), (-1, 1))
    p = np.reshape(a - np.dot(U, m), (-1, 1))
    Ra = np.linalg.norm(p)
    P = np.reshape(np.multiply((1 / Ra), p), (-1, 1))
    n = np.reshape(np.dot(np.transpose(V), b), (-1, 1))
```

```
q = b - np.dot(V, n)
Rb = np.linalg.norm(q)
Q = np.reshape(np.multiply((1 / Rb), q), (-1, 1))

k = S
K = np.zeros((k.shape[0] + 1, k.shape[0] + 1))
K[:-1,:-1] = k
stack = np.vstack(np.append(m, Ra))
t = np.reshape(np.append(n, Rb), (1, -1))
dot = np.dot(stack, t)
K = np.add(K, dot)

D, P = np.linalg.eig(K)

return (np.transpose(np.linalg.inv(P)), np.diag(D), P)
```