# Updates to “rdrobust” package Python (v.1.0.6), August, 2022

1. Definition of dups/dupsid for vce=”nn” (rdrobust/rdbwselect)

###### OLD CODE ################################################

# for i in range(N\_l): dups\_l[i]=sum(X\_l==X\_l[i])

# for i in range(N\_r): dups\_r[i]=sum(X\_r==X\_r[i])

# i = 0

# while i < N\_l:

# dupsid\_l[i:int(i+dups\_l[i])] = np.arange(1,int(dups\_l[i]+1))

# i += dups\_l[i]

# i = 0

# while i < N\_r:

# dupsid\_r[i:int(i+dups\_r[i])] = np.arange(1,int(dups\_r[i]+1))

# i += dups\_r[i]

################################################################

#### NEW code ###########################

aux\_l = pd.DataFrame({'nn\_l': np.ones(N\_l), 'X\_l': X\_l })

dups\_l = aux\_l.groupby('X\_l')['nn\_l'].transform('sum').values.astype(int)

dupsid\_l = aux\_l.groupby('X\_l')['nn\_l'].transform('cumsum').values.astype(int)

aux\_r = pd.DataFrame({'nn\_r': np.ones(N\_r), 'X\_r': X\_r })

dups\_r = aux\_r.groupby('X\_r')['nn\_r'].transform('sum').values.astype(int)

dupsid\_r = aux\_r.groupby('X\_r')['nn\_r'].transform('cumsum').values.astype(int)

#########################################

1. vce=”hc2”, ”hc3” (rdrobust/rdrobust\_bw function)

##### OLD CODE #################################

# hii\_l = nanmat(eN\_l)

# for i in range(eN\_l):

# hii\_l[i] = np.matmul(R\_p\_l[i,:],np.matmul(invG\_p\_l,(R\_p\_l\*W\_h\_l)[i,:]))

# hii\_r = nanmat(eN\_r)

# for i in range(eN\_r):

# hii\_r[i] = np.matmul(R\_p\_r[i,:],np.matmul(invG\_p\_r,(R\_p\_r\*W\_h\_r)[i,:]))

#################################################

######## NEW CODE ###############################

hii\_l = np.sum(np.matmul(R\_p\_l,invG\_p\_l)\*(R\_p\_l\*W\_h\_l), axis = 1)

hii\_r = np.sum(np.matmul(R\_p\_r,invG\_p\_r)\*(R\_p\_r\*W\_h\_r), axis = 1)

# compare\_l = np.allclose(hii\_l,aux\_l,atol=1e-20)

# compare\_r = np.allclose(hii\_r,aux\_r,atol=1e-20)

##################################################

1. Rdplot

####### OLD CODE ############################################

bin\_x\_l = np.zeros(len(x\_l))

bin\_x\_r = np.zeros(len(x\_r))

for k in range(J\_star\_l-1):

bin\_x\_l[np.logical\_and(x\_l>=jumps\_l[k],x\_l<jumps\_l[k+1])] = -J\_star\_l+k

bin\_x\_l[x\_l>=jumps\_l[J\_star\_l-1]] = -1

for k in range(J\_star\_r-1):

bin\_x\_r[np.logical\_and(x\_r>=jumps\_r[k],x\_r<jumps\_r[k+1])] = k+1

bin\_x\_r[x\_r>=jumps\_r[J\_star\_r-1]] = J\_star\_r

rdplot\_mean\_bin\_l = np.zeros(J\_star\_l)

rdplot\_mean\_x\_l = np.zeros(J\_star\_l)

rdplot\_mean\_y\_l = np.zeros(J\_star\_l)

rdplot\_mean\_bin\_r = np.zeros(J\_star\_r)

rdplot\_mean\_x\_r = np.zeros(J\_star\_r)

rdplot\_mean\_y\_r = np.zeros(J\_star\_r)

if covs is not None and covs\_eval=="mean":

dummy\_l = pd.get\_dummies(data=bin\_x\_l, drop\_first=True)

dummy\_l = np.array(dummy\_l).reshape(-1,ncol(dummy\_l))

regressors\_l = np.column\_stack([z\_l,dummy\_l])

covs\_model\_l = LR().fit(regressors\_l,y\_l.reshape(-1,1))

yhatZ\_l = covs\_model\_l.predict(regressors\_l)

dummy\_r = pd.get\_dummies(data=bin\_x\_r, drop\_first=True)

dummy\_r = np.array(dummy\_r).reshape(-1,ncol(dummy\_r))

regressors\_r = np.column\_stack([z\_r,dummy\_r])

covs\_model\_r = LR().fit(regressors\_r,y\_r.reshape(-1,1))

yhatZ\_r = covs\_model\_r.predict(regressors\_r)

def mean(x):

if not np.any(x): return(np.nan)

else: return np.mean(x);

def sd(x):

if np.size(x)<=1: return(np.nan)

else: return np.std(x, ddof = 1);

for k in range(J\_star\_l):

rdplot\_mean\_bin\_l[k] = mean(np.array([jumps\_l[k],jumps\_l[k+1]]))

rdplot\_mean\_x\_l[k] = mean(x\_l[bin\_x\_l==-(k+1)])

rdplot\_mean\_y\_l[k] = mean(y\_l[bin\_x\_l==-(k+1)])

if covs is not None and covs\_eval=="mean":

rdplot\_mean\_y\_l[k] = mean(yhatZ\_l[bin\_x\_l==-(k+1)])

rdplot\_mean\_y\_l = rdplot\_mean\_y\_l[::-1]

rdplot\_mean\_x\_l = rdplot\_mean\_x\_l[::-1]

for k in range(J\_star\_r):

rdplot\_mean\_bin\_r[k] = mean(np.array([jumps\_r[k],jumps\_r[k+1]]))

rdplot\_mean\_x\_r[k] = mean(x\_r[bin\_x\_r==(k+1)])

rdplot\_mean\_y\_r[k] = mean(y\_r[bin\_x\_r==(k+1)])

if covs is not None and covs\_eval=="mean":

rdplot\_mean\_y\_r# = mean(yhatZ\_r[bin\_x\_r==(k+1)])

rdplot\_mean\_bin\_l[J\_star\_l-1] = mean(np.array([jumps\_l[J\_star\_l-1],c]))

rdplot\_mean\_bin\_r[J\_star\_r-1] = mean(np.array([jumps\_r[J\_star\_r-1],x\_max]))

bin\_x = np.concatenate([bin\_x\_l,bin\_x\_r])

rdplot\_mean\_bin = np.concatenate([rdplot\_mean\_bin\_l,rdplot\_mean\_bin\_r])

rdplot\_mean\_x = np.concatenate([rdplot\_mean\_x\_l,rdplot\_mean\_x\_r])

rdplot\_mean\_y = np.concatenate([rdplot\_mean\_y\_l,rdplot\_mean\_y\_r])

rdplot\_sd\_y\_l = np.zeros(J\_star\_l)##

rdplot\_N\_l = np.zeros(J\_star\_l)

for j in range(J\_star\_l):

rdplot\_sd\_y\_l[j] = sd(y\_l[bin\_x\_l==-(j+1)])

rdplot\_N\_l[j] = len(y\_l[bin\_x\_l==-(j+1)])

rdplot\_sd\_y\_r = np.zeros(J\_star\_r)

rdplot\_N\_r = np.zeros(J\_star\_r)

for j in range(J\_star\_r):

rdplot\_sd\_y\_r[j] = sd(y\_r[bin\_x\_r==(j+1)])

rdplot\_N\_r[j] = len(y\_r[bin\_x\_r==(j+1)])

############ END OLD CODE ##########################################

############ NEW CODE ##############################################

bin\_x\_l = np.searchsorted(jumps\_l, x\_l,side='right') - J\_star\_l - 1

bin\_x\_r = np.searchsorted(jumps\_r, x\_r,side='left')

# compare\_bin\_x\_l = np.allclose(bin\_x\_l, bin\_x\_l2)

# compare\_bin\_x\_r = np.allclose(bin\_x\_r, bin\_x\_r2)

aux\_l = pd.DataFrame({'bin\_x\_l':bin\_x\_l, 'y\_l':y\_l, 'x\_l':x\_l})

rdplot\_l = aux\_l.groupby('bin\_x\_l').agg({'y\_l': 'mean', 'x\_l':'mean'}).reset\_index()

rdplot\_bin\_l = rdplot\_l['bin\_x\_l'].values

rdplot\_mean\_y\_l = rdplot\_l['y\_l'].values

rdplot\_mean\_x\_l = rdplot\_l['x\_l'].values

# compare\_y\_l = np.allclose(rdplot\_mean\_y\_l2, rdplot\_mean\_y\_l)

# compare\_x\_l = np.allclose(rdplot\_mean\_x\_l2, rdplot\_mean\_x\_l)

aux\_r = pd.DataFrame({'bin\_x\_r':bin\_x\_r, 'y\_r':y\_r, 'x\_r':x\_r})

rdplot\_r = aux\_r.groupby('bin\_x\_r').agg({'y\_r': 'mean', 'x\_r':'mean'}).reset\_index()

rdplot\_bin\_r = rdplot\_r['bin\_x\_r'].values # Only 34 values (insteaf of 35), bin 29 is missing?!?!

rdplot\_mean\_y\_r = rdplot\_r['y\_r'].values

rdplot\_mean\_x\_r = rdplot\_r['x\_r'].values

# compare\_y\_r = np.allclose(rdplot\_mean\_y\_r2, rdplot\_mean\_y\_r)

# compare\_x\_r = np.allclose(rdplot\_mean\_x\_r2, rdplot\_mean\_x\_r)

t\_ind\_l = np.arange(J\_star\_l)

t\_ind\_r = np.arange(J\_star\_r)

rdplot\_mean\_bin\_l = np.mean(np.column\_stack((jumps\_l[t\_ind\_l],jumps\_l[t\_ind\_l+1])),axis=1)

rdplot\_mean\_bin\_r = np.mean(np.column\_stack((jumps\_r[t\_ind\_r],jumps\_r[t\_ind\_r+1])),axis=1)

rdplot\_mean\_bin\_l = rdplot\_mean\_bin\_l[np.flip(-rdplot\_bin\_l)-1]

rdplot\_mean\_bin\_r = rdplot\_mean\_bin\_r[rdplot\_bin\_r-1]

# compare\_l = np.allclose(rdplot\_mean\_bin\_l, rdplot\_mean\_bin\_l2)

# compare\_r = np.allclose(rdplot\_mean\_bin\_r, rdplot\_mean\_bin\_r2)

bin\_x = np.concatenate((bin\_x\_l,bin\_x\_r))

rdplot\_mean\_bin = np.concatenate((rdplot\_mean\_bin\_l, rdplot\_mean\_bin\_r))

rdplot\_mean\_x = np.concatenate((rdplot\_mean\_x\_l, rdplot\_mean\_x\_r))

rdplot\_mean\_y = np.concatenate((rdplot\_mean\_y\_l, rdplot\_mean\_y\_r))

# compare1 = np.allclose(rdplot\_mean\_bin, rdplot\_mean\_bin2)

# compare2 = np.allclose(rdplot\_mean\_x, rdplot\_mean\_x2)

# compare3 = np.allclose(rdplot\_mean\_y, rdplot\_mean\_y2)

rdplot\_mean\_bin = np.concatenate((rdplot\_mean\_bin\_l, rdplot\_mean\_bin\_r))

rdplot\_mean\_x = np.concatenate((rdplot\_mean\_x\_l, rdplot\_mean\_x\_r))

rdplot\_mean\_y = np.concatenate((rdplot\_mean\_y\_l, rdplot\_mean\_y\_r))

aux\_l = pd.DataFrame({'bin':-bin\_x\_l, 'y':y\_l}).groupby('bin').agg({'y':['count','std']})

aux\_r = pd.DataFrame({'bin':bin\_x\_r, 'y':y\_r}).groupby('bin').agg({'y':['count','std']})

rdplot\_N\_l = aux\_l['y']['count'].values

rdplot\_sd\_y\_l = aux\_l['y']['std'].values

rdplot\_N\_r = aux\_r['y']['count'].values

rdplot\_sd\_y\_r = aux\_r['y']['std'].values

####### OLD CODE ################################

# cutoffs = np.concatenate([jumps\_l,jumps\_r[1:]])

# rdplot\_min\_bin = cutoffs[:-1]

# rdplot\_max\_bin = cutoffs[1:]

#################################################

####### NEW CODE ################################

rdplot\_min\_bin\_l = jumps\_l[0:J\_star\_l]

rdplot\_max\_bin\_l = jumps\_l[1:(J\_star\_l + 1)]

rdplot\_min\_bin\_r = jumps\_r[0:J\_star\_r]

rdplot\_max\_bin\_r = jumps\_r[1:(J\_star\_r + 1)]

rdplot\_min\_bin = np.concatenate((rdplot\_min\_bin\_l[np.flip(-rdplot\_bin\_l)-1], rdplot\_min\_bin\_r[rdplot\_bin\_r-1]))

rdplot\_max\_bin = np.concatenate((rdplot\_max\_bin\_l[np.flip(-rdplot\_bin\_l)-1], rdplot\_max\_bin\_r[rdplot\_bin\_r-1]))

####################################################

1. rdrobust\_vce function

def rdrobust\_vce(d, s, RX, res, C):

########### OLD CODE ################################

# k = ncol(RX)

# M = np.zeros((k,k))

# if C is None:

# w = 1

# if d==0:

# M = crossprod(res\*RX,res\*RX)

# else:

# for i in range(d+1):

# SS = (res[:,i].reshape(-1,1))\*res

# for j in range(d+1):

# M += crossprod(RX\*(s[i]\*s[j])\*SS[:,j].reshape(-1,1),RX)

# else:

# try: n = len(C)

# except: n = 1

# clusters = np.unique(C)

# g = len(clusters)

# w = ((n-1)/(n-k))\*(g/(g-1))

# if d==0:

# for i in range(g):

# ind = C==clusters[i]

# Xi = RX[ind,:]

# ri = res[ind,:]

# Xiri = crossprod(Xi,ri).T

# M = M + crossprod(Xiri,Xiri)

# else:

# for i in range(g):

# ind = C==clusters[i]

# Xi = RX[ind,:]

# ri = res[ind,:]

# for l in range(d+1):

# for j in range(d+1):

# M = M + crossprod(crossprod(Xi,s[l]\*ri[:,l]).T,crossprod(Xi,s[j]\*ri[:,j]).T)

####### END OLD CODE ###############################################

###### NEW CODE #################################################

k = ncol(RX)

M = np.zeros((k,k))

if C is None:

w = 1

if d==0:

M = crossprod(res\*RX,res\*RX)

else:

for i in range(d+1):

SS = (res[:,i].reshape(-1,1))\*res

for j in range(d+1):

M += crossprod(RX\*(s[i]\*s[j])\*SS[:,j].reshape(-1,1),RX)

else:

try: n = len(C)

except: n = 1

clusters = np.unique(C)

g = len(clusters)

w = ((n-1)/(n-k))\*(g/(g-1))

if d==0:

for i in range(g):

ind = C==clusters[i]

Xi = RX[ind,:]

ri = res[ind,:]

Xr = crossprod(Xi,ri).T

M = M + crossprod(Xr,Xr)

else:

for i in range(g):

ind = C==clusters[i]

Xi = RX[ind,:]

ri = res[ind,:]

MHolder = np.zeros((1+d,k))

for l in range(d+1):

MHolder[l,:] = crossprod(Xi,s[l]\*ri[:,l]).T

summedvalues = np.sum(MHolder, axis = 0).T

M = M + crossprod(summedvalues,summedvalues)

return w\*M;

1. Class .self in funs.py
2. else:
3. print(self.coef.index[j].ljust(fw\_l),
4. ############ OLD CODE #######################
5. # '-'.rjust(fw\_c),
6. # '-'.rjust(fw\_c),
7. ###############################################
8. ########### NEW CODE ##########################
9. str(round(float(self.coef.iloc[j]),n\_dec)).rjust(fw\_c),
10. str(round(float(self.se.iloc[j]),n\_dec)).rjust(fw\_c),
11. ###############################################