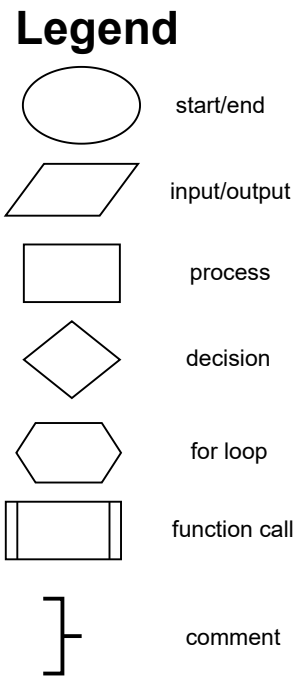
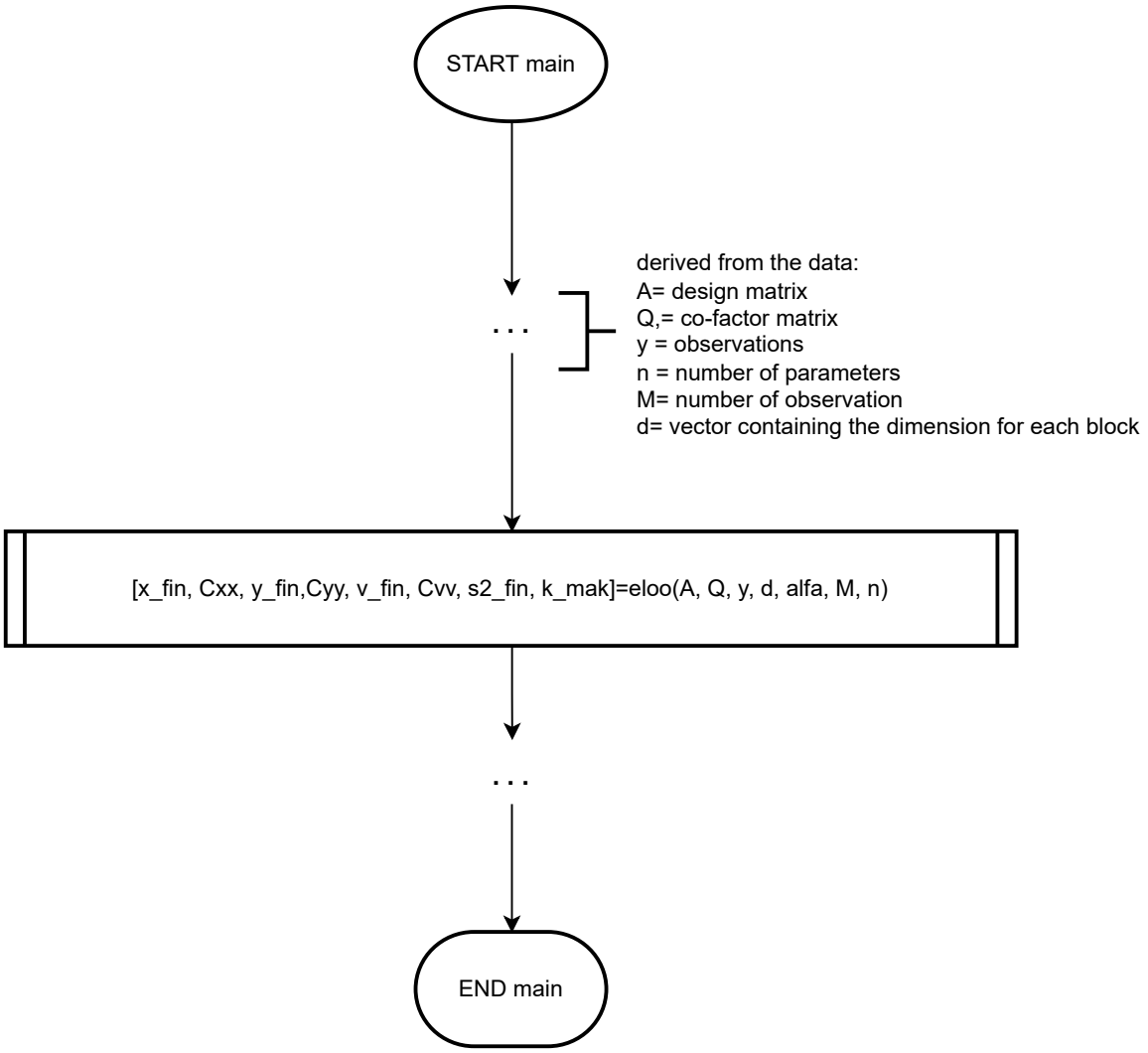
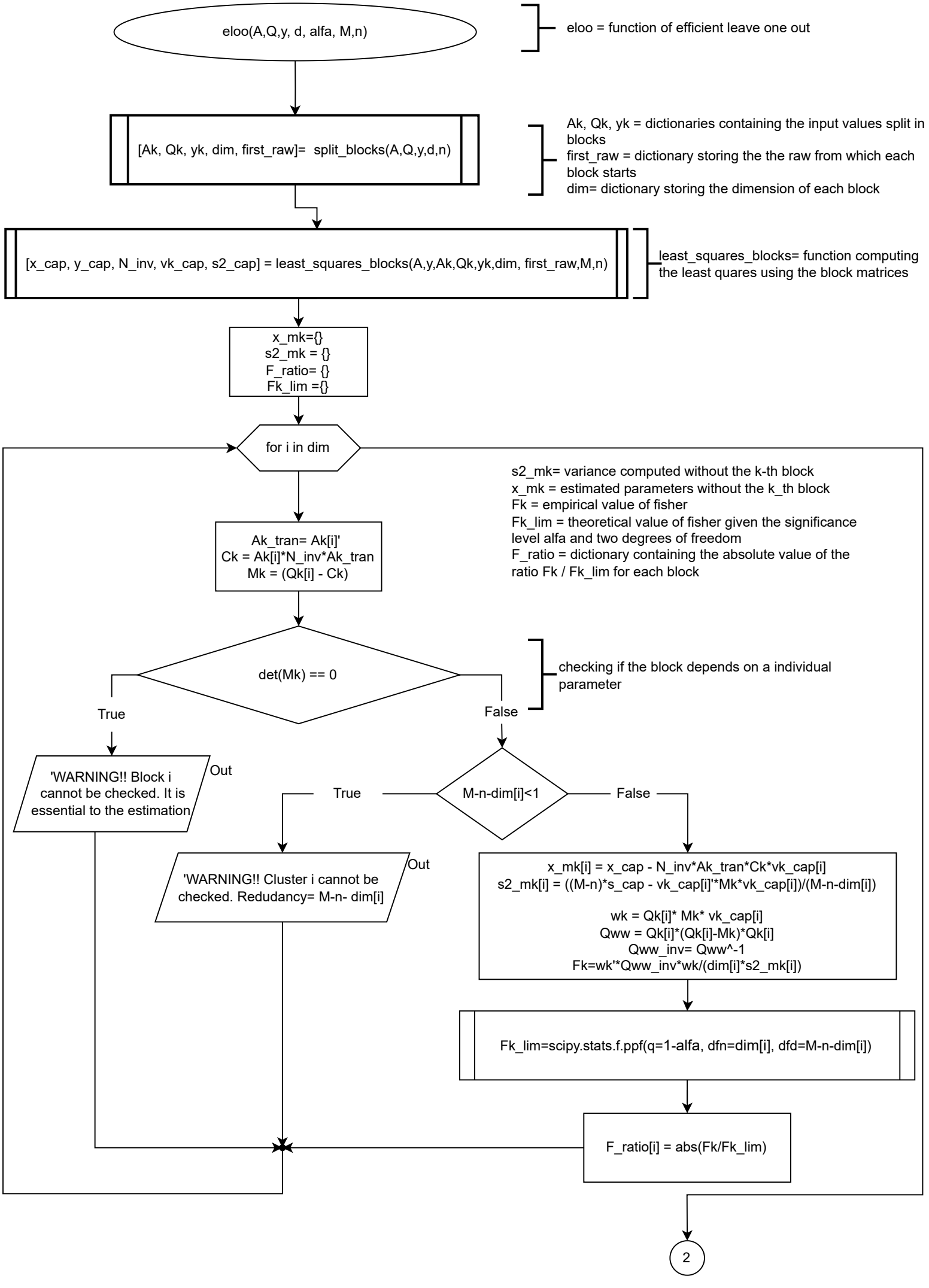
ELOBO flowchart: Efficient Leave One Block Out

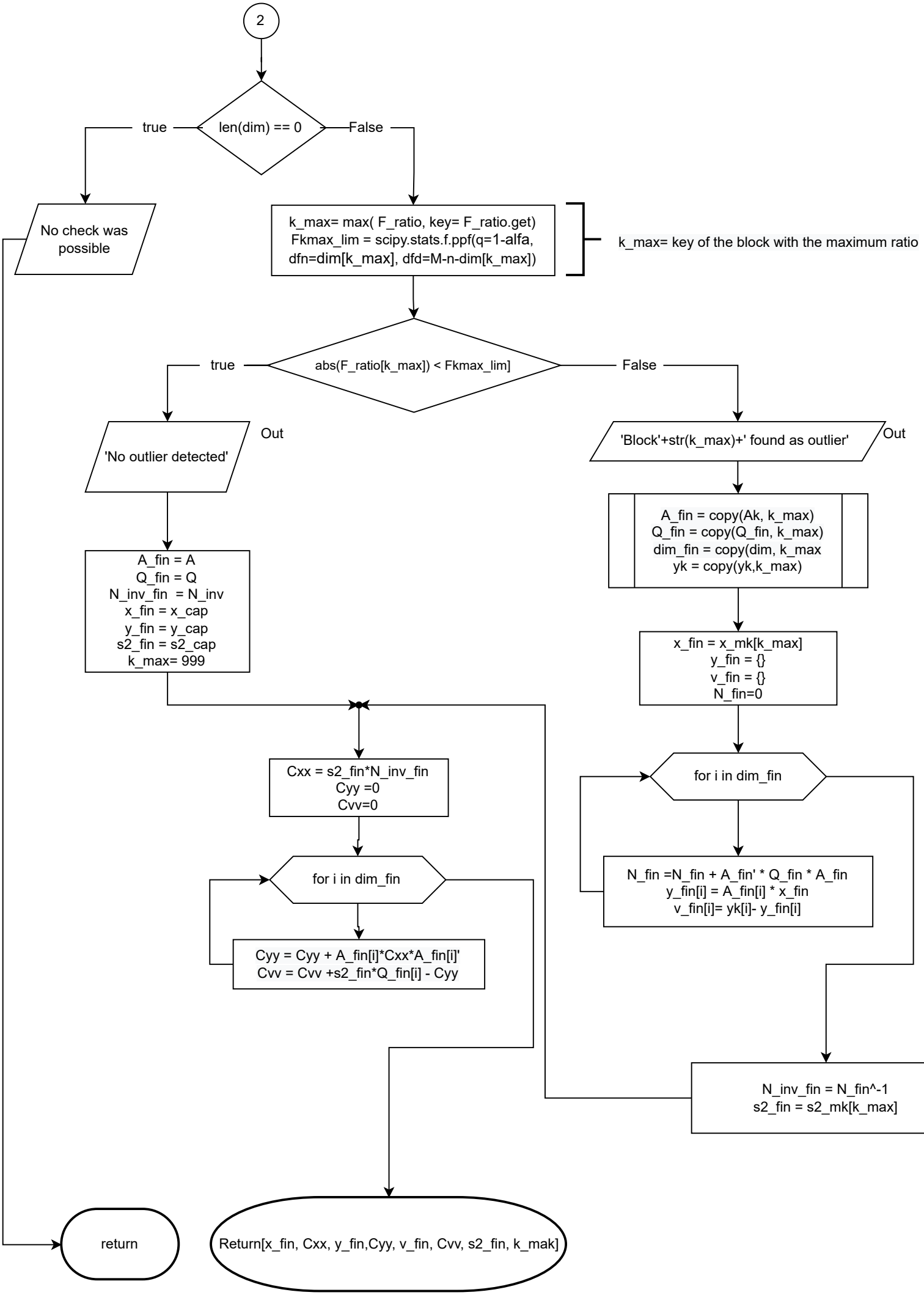
The functions represented are 'elobo' (main function implementing the algorithm) which calls 'split_blocks (splits input arrays and vectors into blocks by saving them in dictionaries) and 'least_squares_block (implements least squares procedures using block decomposition). Elobo also calls 'copy' which returns a copy of a dictionary with a specific element deleted; in this way the original dictionary does not change.
Also 'lobo' function is described (implement the classic leave one block out in wich you repeat the least square procedure reomving each time a block).

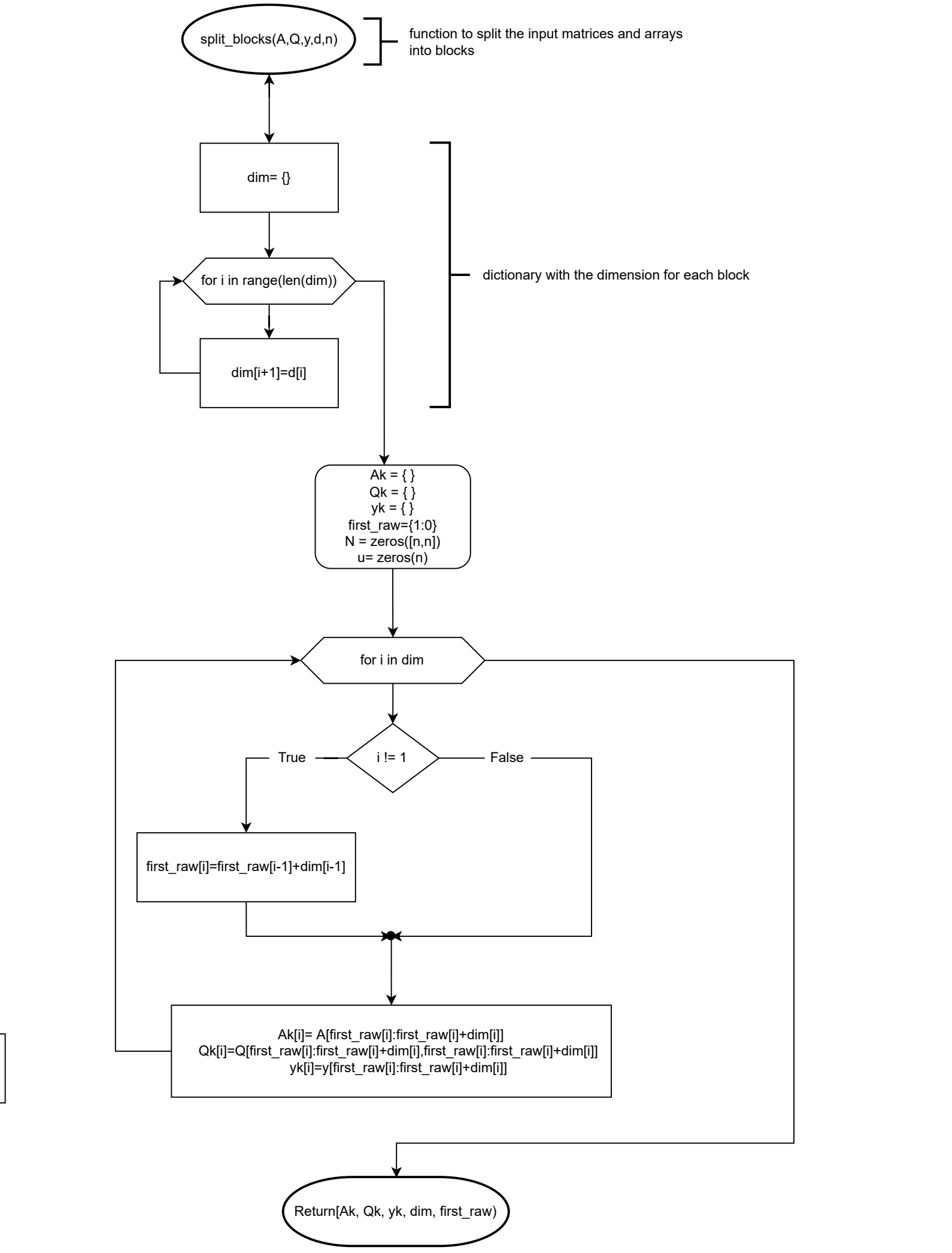
In this diagram operations on arrays are not explicit with a python function, but they will be handled with numpy library.
The structure of the dictionary will be used to manage the subdivision into blocks; the key-value structure in my opinion is more intuitive to access to a specific element

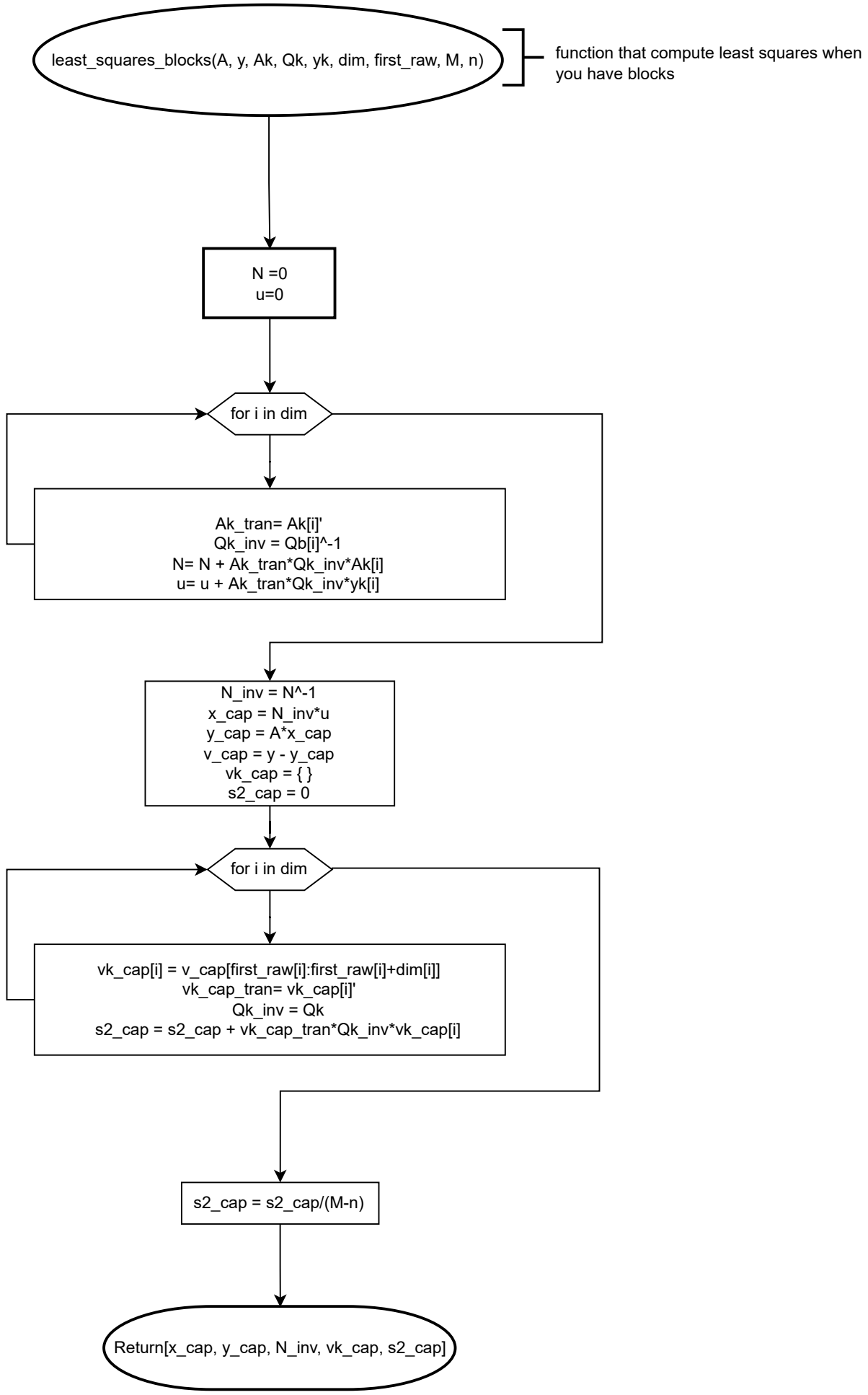


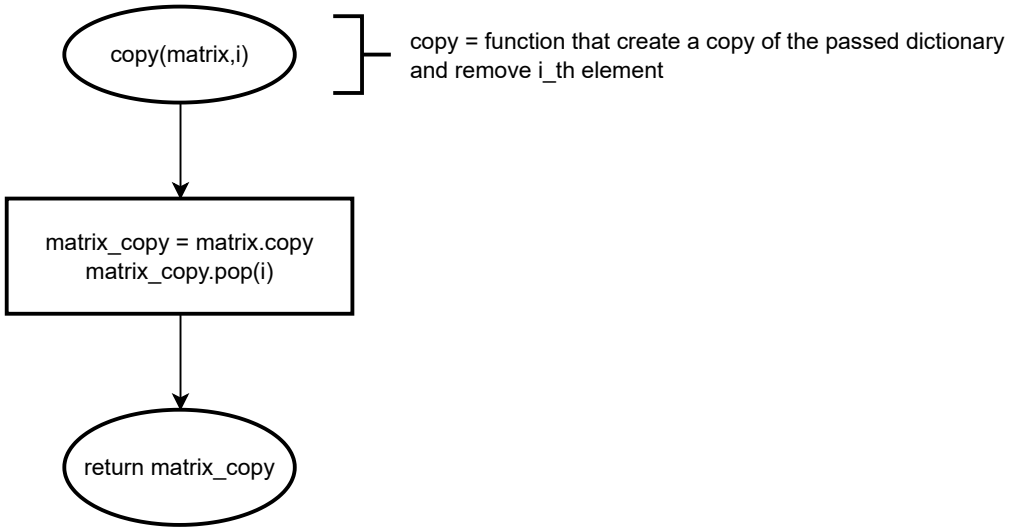

START main
. . .
derived from the data:
A= design matrix
Q,= co-factor matrix
y = observations
n = number of parameters
M= number of observation
d= vector containing the dimension for each block
[x_fin, Cxx, y_fin,Cyy, v_fin, Cvv, s2_fin, k_mak]=eloo(A, Q, y, d, alfa, M, n)
. . .
END main

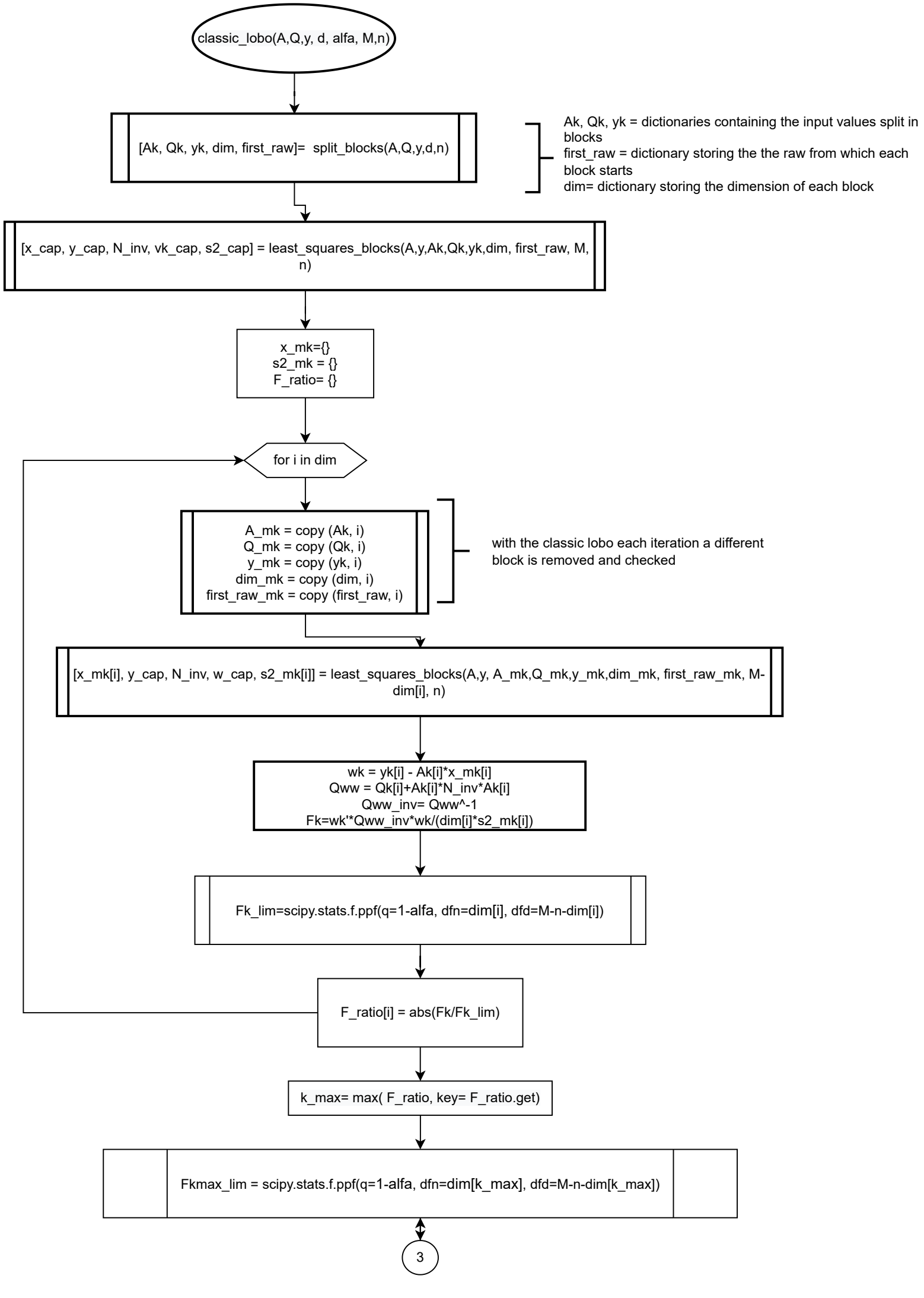

eloo(A,Q,y, d, alfa, M,n)
eloo = function of efficient leave one out
[Ak, Qk, yk, dim, first_raw]= split_blocks(A,Q,y,d,n)
Ak, Qk, yk = dictionaries containing the input values split in blocks
first_raw = dictionary storing the the raw from which each block starts
dim= dictionary storing the dimension of each block
[x_cap, y_cap, N_inv, vk_cap, s2_cap] = least_squares_blocks(A,y,Ak,Qk,yk,dim, first_raw,M,n)
least_squares_blocks= function computing the least quares using the block matrices
x_mk={}
s2_mk = {}
F_ratio= {}
Fk_lim ={}
for i in dim
s2_mk= variance computed without the k-th block
x_mk = estimated parameters without the k_th block
Fk = empirical value of fisher
Fk_lim = theoretical value of fisher given the significance level alfa and two degrees of freedom
F_ratio = dictionary containing the absolute value of the ratio Fk / Fk_lim for each block
Ak_tran= Ak[i]'
Ck = Ak[i]*N_inv*Ak_tran
Mk = (Qk[i] - Ck)
det(Mk) == 0
checking if the block depends on a individual parameter
True
False
'WARNING!! Block i cannot be checked. It is essential to the estimation
Out
M-n-dim[i]<1
True
False
'WARNING!! Cluster i cannot be checked. Redudancy= M-n- dim[i]
Out
x_mk[i] = x_cap - N_inv*Ak_tran*Ck*vk_cap[i]
s2_mk[i] = ((M-n)*s_cap - vk_cap[i]'*Mk*vk_cap[i])/(M-n-dim[i])
wk = Qk[i]* Mk* vk_cap[i]
Qww = Qk[i]*(Qk[i]-Mk)*Qk[i]
Qww_inv= Qww^-1
Fk=wk'*Qww_inv*wk/(dim[i]*s2_mk[i])
Fk_lim=scipy.stats.f.ppf(q=1-alfa, dfn=dim[i], dfd=M-n-dim[i])
F_ratio[i] = abs(Fk/Fk_lim)
2

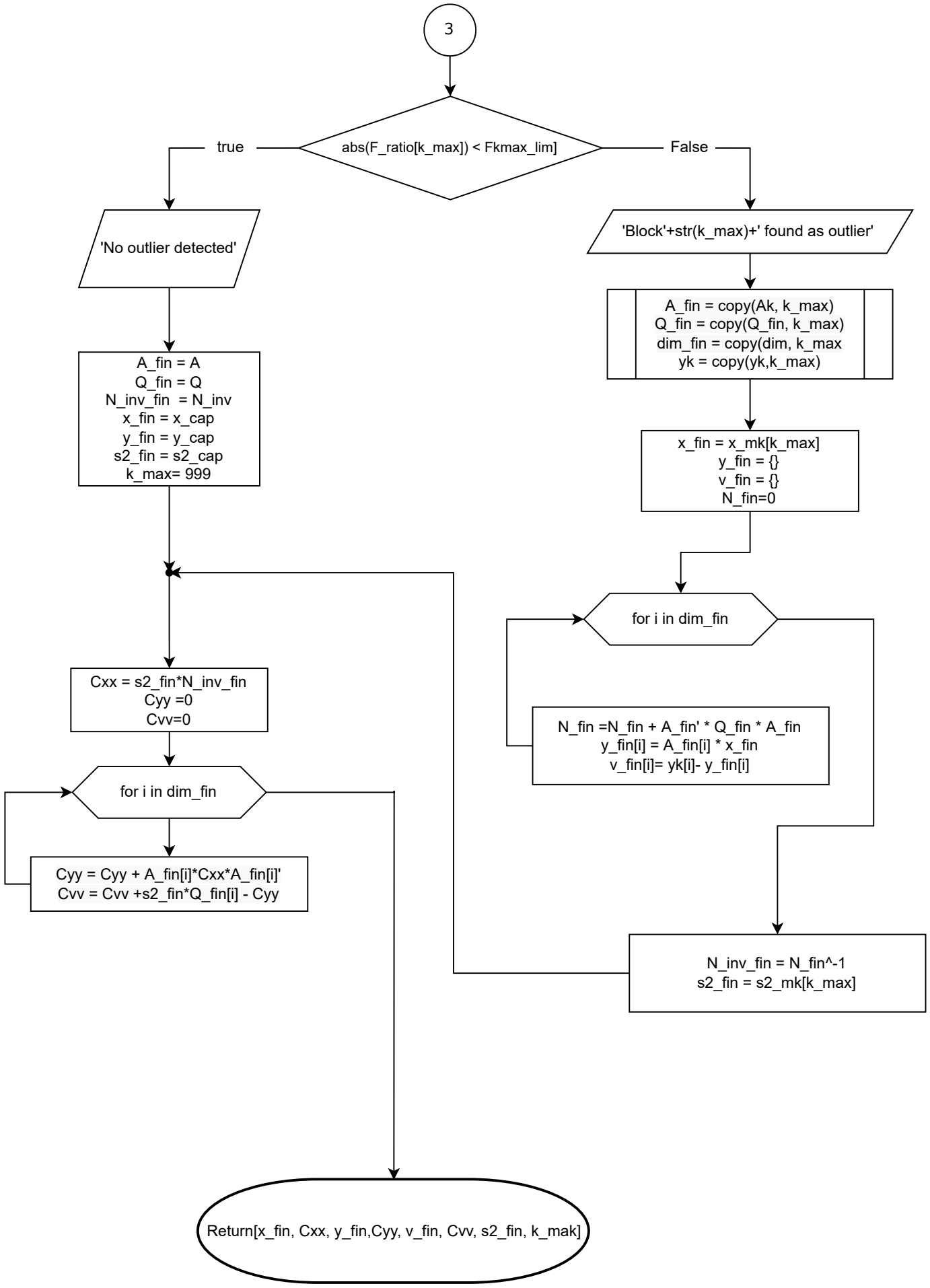

2
len(dim) == 0
true
False
No check was possible
k_max= max(F_ratio, key= F_ratio.get)
Fkmax_lim = scipy.stats.f.ppf(q=1-alfa, dfn=dim[k_max], dfd=M-n-dim[k_max])
k_max= key of the block with the maximum ratio
abs(F_ratio[k_max]) < Fkmax_lim]
true
False
'No outlier detected'
Out
'Block'+str(k_max)+' found as outlier'
Out
A_fin = A
Q_fin = Q
N_inv_fin = N_inv
x_fin = x_cap
y_fin = y_cap
s2_fin = s2_cap
k_max= 999
A_fin = copy(Ak, k_max)
Q_fin = copy(Q_fin, k_max)
dim_fin = copy(dim, k_max
yk = copy(yk,k_max)
x_fin = x_mk[k_max]
y_fin = {}
v_fin = {}
N_fin=0
Cxx = s2_fin*N_inv_fin
Cyy =0
Cvv=0
for i in dim_fin
Cyy = Cyy + A_fin[i]*Cxx*A_fin[i]'
Cvv = Cvv +s2_fin*Q_fin[i] - Cyy
for i in dim_fin
N_fin =N_fin + A_fin' * Q_fin * A_fin
y_fin[i] = A_fin[i] * x_fin
v_fin[i]= yk[i]- y_fin[i]
N_inv_fin = N_fin^-1
s2_fin = s2_mk[k_max]
return
Return[x_fin, Cxx, y_fin,Cyy, v_fin, Cvv, s2_fin, k_mak]


split_blocks(A,Q,y,d,n)
function to split the input matrices and arrays into blocks
dim= {}
for i in range(len(dim))
dim[i+1]=d[i]
dictionary with the dimension for each block
Ak = { }
Qk = { }
yk = { }
first_raw={1:0}
N = zeros([n,n])
u= zeros(n)
for i in dim
i != 1
True
False
first_raw[i]=first_raw[i-1]+dim[i-1]
Ak[i]= A[first_raw[i]:first_raw[i]+dim[i]]
Qk[i]=Q[first_raw[i]:first_raw[i]+dim[i],first_raw[i]:first_raw[i]+dim[i]]
yk[i]=y[first_raw[i]:first_raw[i]+dim[i]]
Return[Ak, Qk, yk, dim, first_raw)


least_squares_blocks(A, y, Ak, Qk, yk, dim, first_raw, M, n)
function that compute least squares when you have blocks
N =0
u=0
for i in dim
Ak_tran= Ak[i]'
Qk_inv = Qb[i]^-1
N= N + Ak_tran*Qk_inv*Ak[i]
u= u + Ak_tran*Qk_inv*yk[i]
N_inv = N^-1
x_cap = N_inv*u
y_cap = A*x_cap
v_cap = y - y_cap
vk_cap = { }
s2_cap = 0
for i in dim
vk_cap[i] = v_cap[first_raw[i]:first_raw[i]+dim[i]]
vk_cap_tran= vk_cap[i]'
Qk_inv = Qk
s2_cap = s2_cap + vk_cap_tran*Qk_inv*vk_cap[i]
s2_cap = s2_cap/(M-n)
Return[x_cap, y_cap, N_inv, vk_cap, s2_cap]


copy(matrix,i)
copy = function that create a copy of the passed dictionary and remove i_th element
matrix_copy = matrix.copy
matrix_copy.pop(i)
return matrix_copy


classic_lobo(A,Q,y, d, alfa, M,n)
[Ak, Qk, yk, dim, first_raw]= split_blocks(A,Q,y,d,n)
Ak, Qk, yk = dictionaries containing the input values split in blocks
first_raw = dictionary storing the the raw from which each block starts
dim= dictionary storing the dimension of each block
[x_cap, y_cap, N_inv, vk_cap, s2_cap] = least_squares_blocks(A,y,Ak,Qk,yk,dim, first_raw, M, n)
x_mk={}
s2_mk = {}
F_ratio= {}
for i in dim
A_mk = copy (Ak, i)
Q_mk = copy (Qk, i)
y_mk = copy (yk, i)
dim_mk = copy (dim, i)
first_raw_mk = copy (first_raw, i)
with the classic lobo each iteration a different block is removed and checked
[x_mk[i], y_cap, N_inv, w_cap, s2_mk[i]] = least_squares_blocks(A,y, A_mk,Q_mk,y_mk,dim_mk, first_raw_mk, M-dim[i], n)
wk = yk[i] - Ak[i]*x_mk[i]
Qww = Qk[i]+Ak[i]*N_inv*Ak[i]
Qww_inv= Qww^-1
Fk=wk'*Qww_inv*wk/(dim[i]*s2_mk[i])
Fk_lim=scipy.stats.f.ppf(q=1-alfa, dfn=dim[i], dfd=M-n-dim[i])
F_ratio[i] = abs(Fk/Fk_lim)
k_max= max(F_ratio, key= F_ratio.get)
Fkmax_lim = scipy.stats.f.ppf(q=1-alfa, dfn=dim[k_max], dfd=M-n-dim[k_max])
3


3
abs(F_ratio[k_max]) < Fkmax_lim]
true
False
'No outlier detected'
'Block'+str(k_max)+' found as outlier'
A_fin = A
Q_fin = Q
N_inv_fin = N_inv
x_fin = x_cap
y_fin = y_cap
s2_fin = s2_cap
k_max= 999
A_fin = copy(Ak, k_max)
Q_fin = copy(Q_fin, k_max)
dim_fin = copy(dim, k_max
yk = copy(yk,k_max)
x_fin = x_mk[k_max]
y_fin = {}
v_fin = {}
N_fin=0
for i in dim_fin
N_fin =N_fin + A_fin' * Q_fin * A_fin
y_fin[i] = A_fin[i] * x_fin
v_fin[i]= yk[i]- y_fin[i]
Cxx = s2_fin*N_inv_fin
Cyy =0
Cvv=0
for i in dim_fin
Cyy = Cyy + A_fin[i]*Cxx*A_fin[i]'
Cvv = Cvv +s2_fin*Q_fin[i] - Cyy
N_inv_fin = N_fin^-1
s2_fin = s2_mk[k_max]
Return[x_fin, Cxx, y_fin,Cyy, v_fin, Cvv, s2_fin, k_mak]