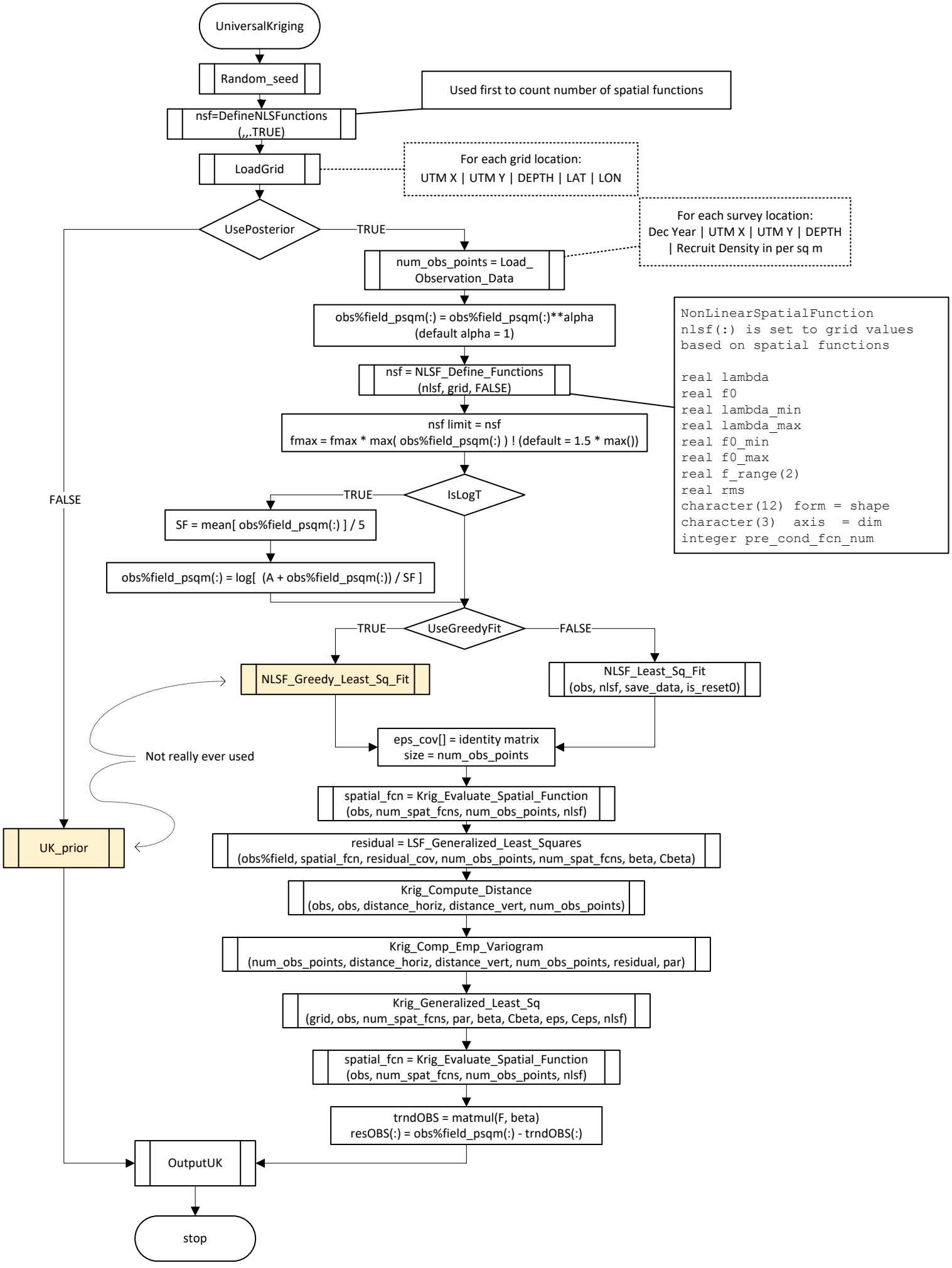
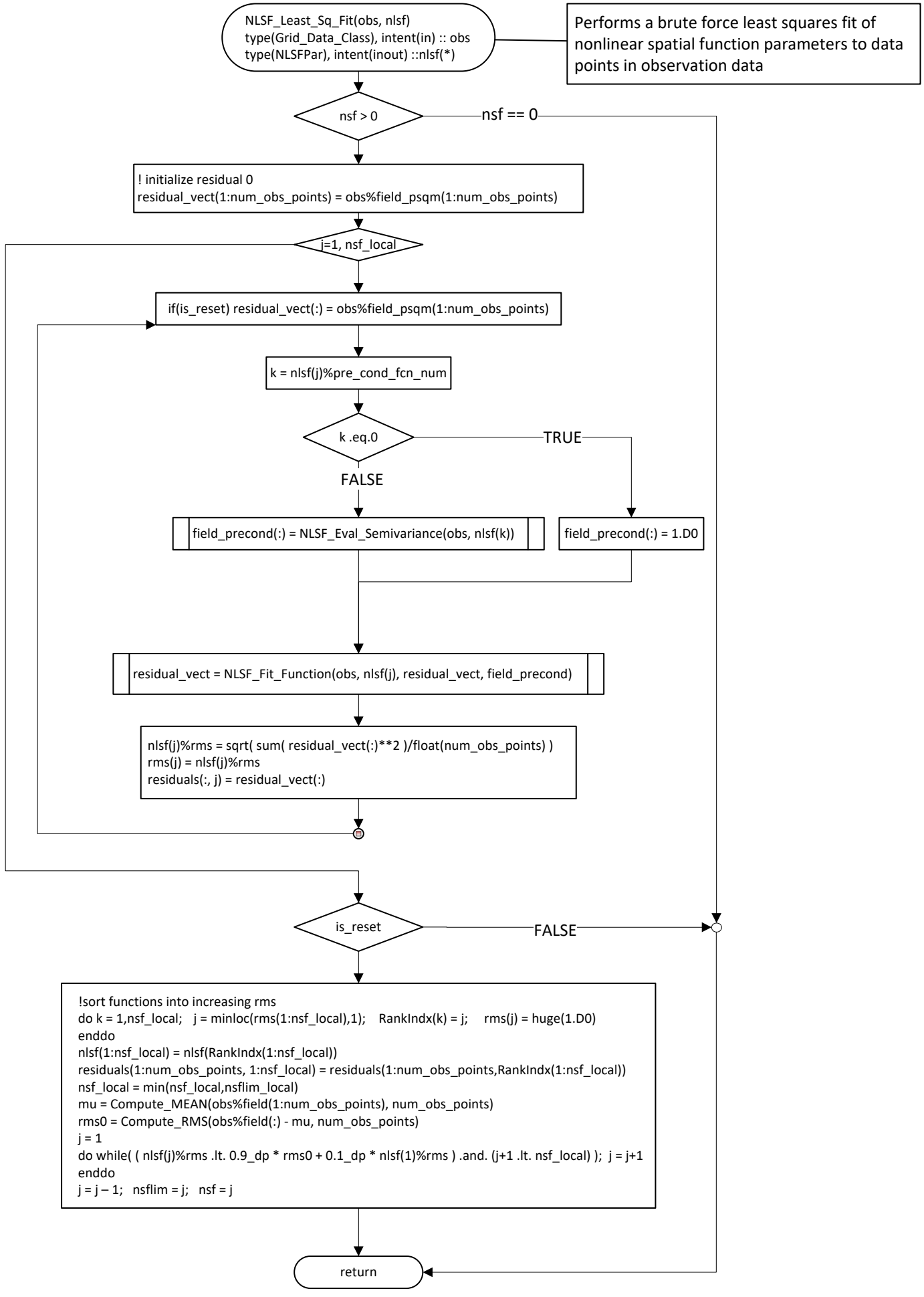
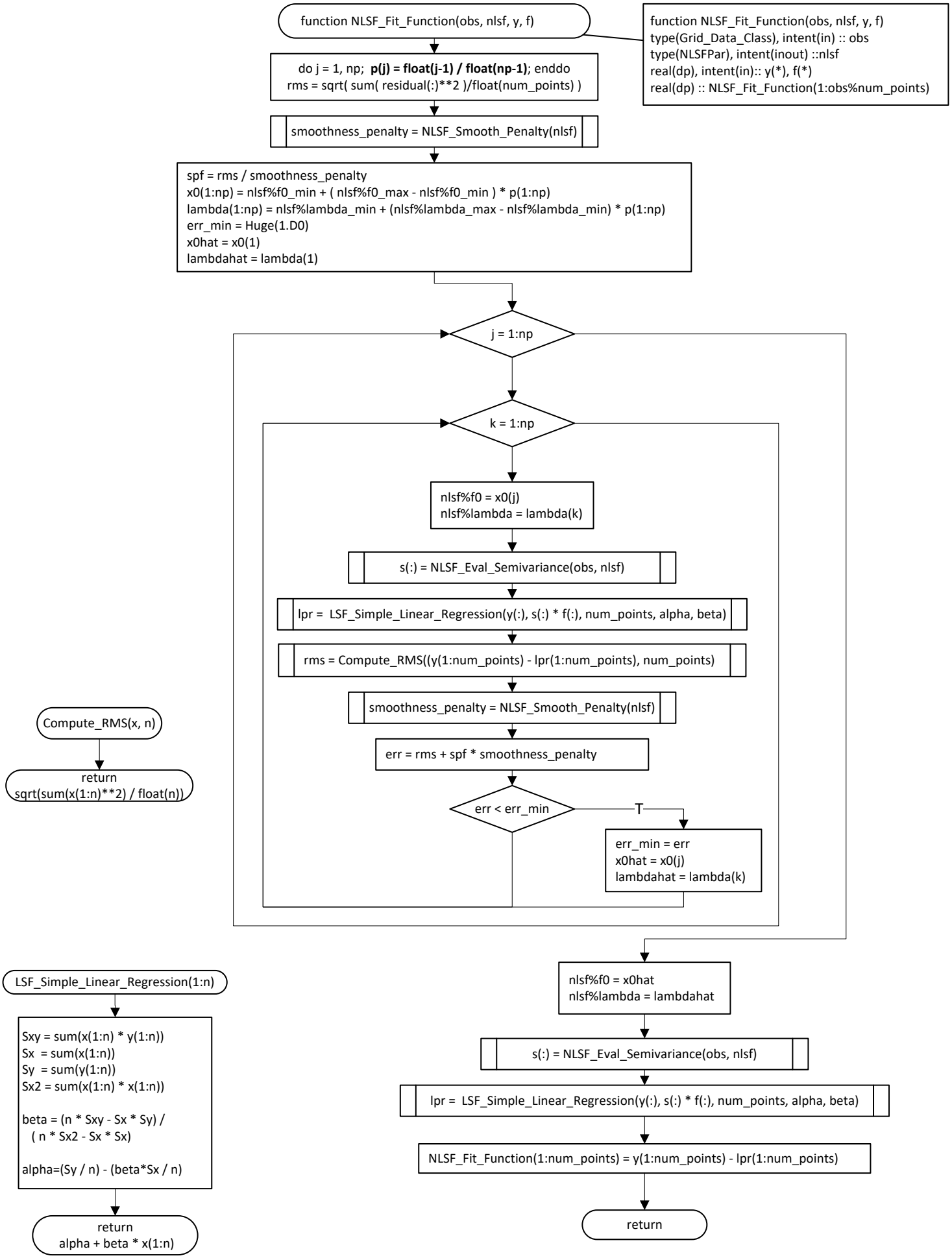
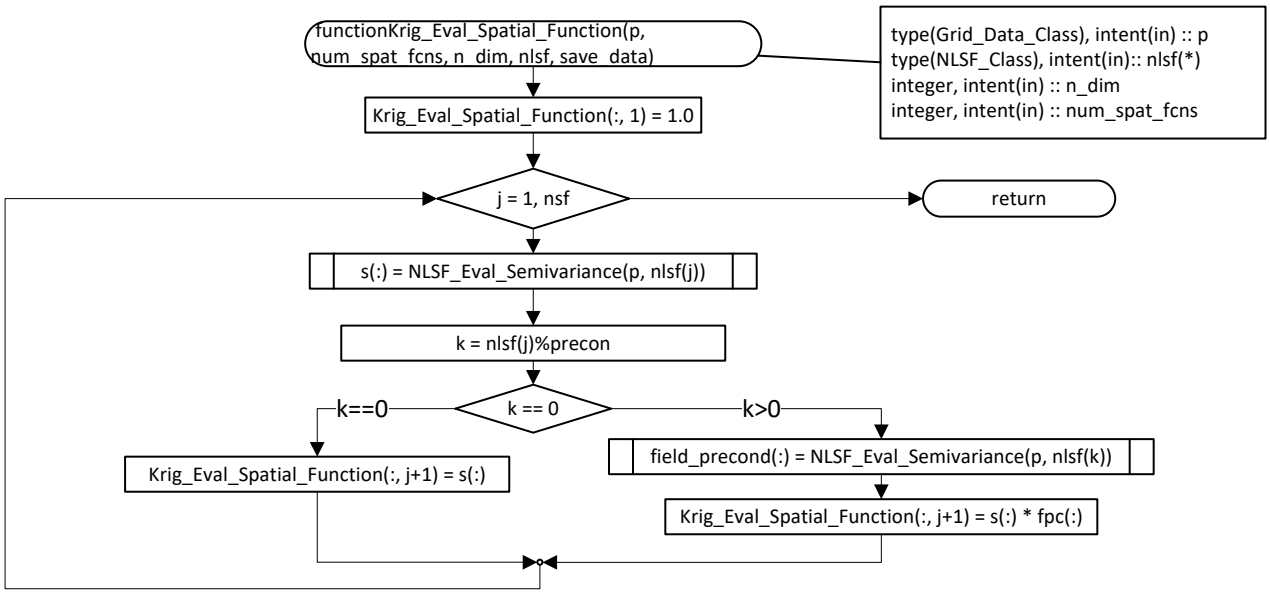
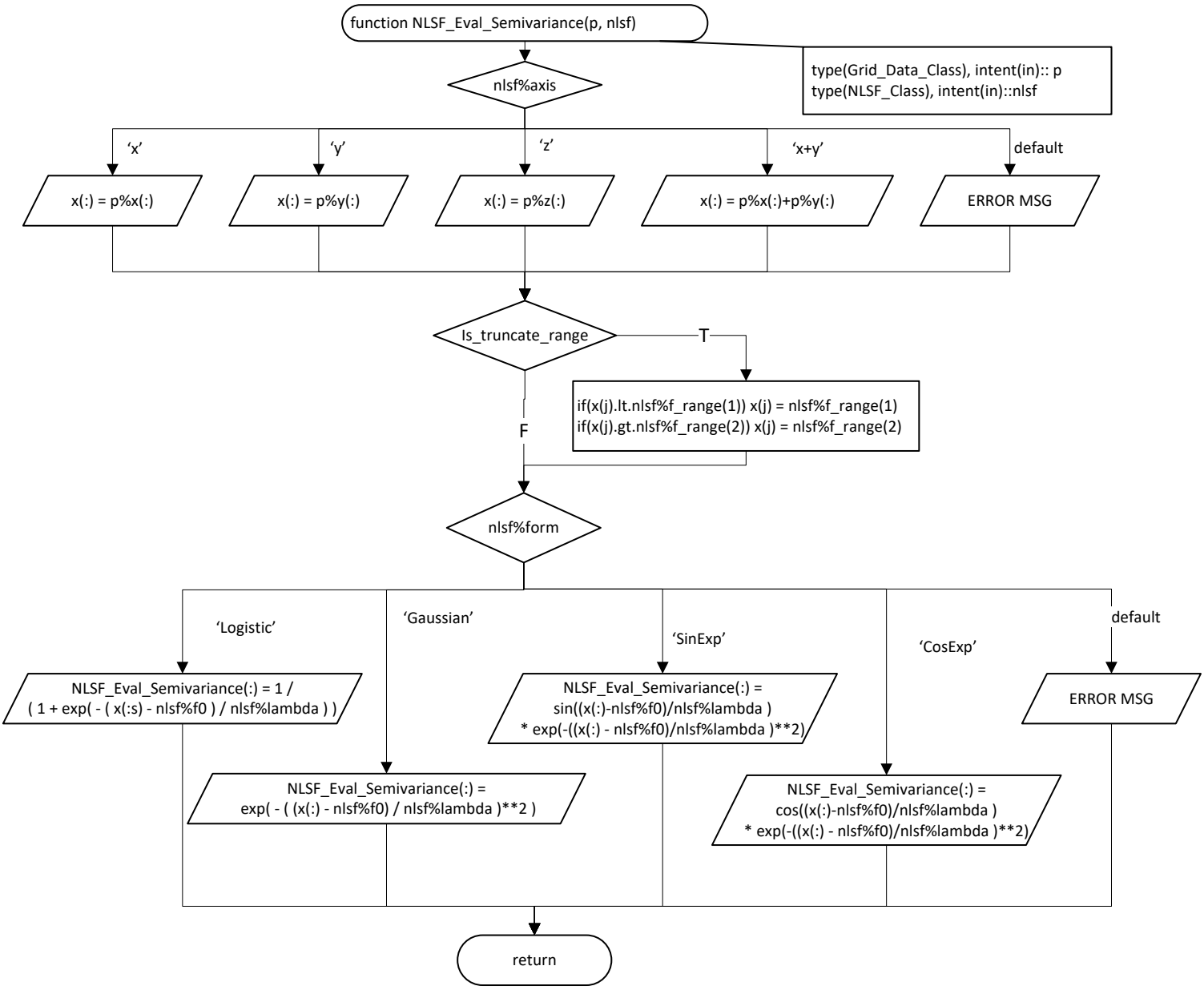

UniversalKriging
Random_seed
nsf=DefineNLSFunctions (,,.TRUE)
Used first to count number of spatial functions
LoadGrid
For each grid location: UTM X | UTM Y | DEPTH | LAT | LON
UsePosterior
TRUE
FALSE
For each survey location: Dec Year | UTM X | UTM Y | DEPTH | Recruit Density in per sq m
num_obs_points = Load_ Observation_Data
obs%field_psqm(:) = obs%field_psqm(:)**alpha (default alpha = 1)
nsf = NLSF_Define_Functions (nlsf, grid, FALSE)
NonLinearSpatialFunction nlsf(:) is set to grid values based on spatial functions
real lambda
real f0
real lambda_min
real lambda_max
real f0_min
real f0_max
real f_range(2)
real rms
character(12) form = shape
character(3) axis = dim
integer pre_cond_fcn_num
nsf limit = nsf
fmax = fmax * max(obs%field_psqm(:)) ! (default = 1.5 * max())
IsLogT
TRUE
SF = mean[obs%field_psqm(:)] / 5
obs%field_psqm(:) = log[(A + obs%field_psqm(:)) / SF]
UseGreedyFit
TRUE
FALSE
NLSF_Greedy_Least_Sq_Fit
NLSF_Least_Sq_Fit (obs, nlsf, save_data, is_reset0)
eps_cov[] = identity matrix size = num_obs_points
Not really ever used
spatial_fcn = Krig_Evaluate_Spatial_Function (obs, num_spat_fcns, num_obs_points, nlsf)
UK_prior
residual = LSF_Generalized_Least_Squares (obs%field, spatial_fcn, residual_cov, num_obs_points, num_spat_fcns, beta, Cbeta)
Krig_Compute_Distance (obs, obs, distance_horiz, distance_vert, num_obs_points)
Krig_Comp_Emp_Variogram (num_obs_points, distance_horiz, distance_vert, num_obs_points, residual, par)
Krig_Generalized_Least_Sq (grid, obs, num_spat_fcns, par, beta, Cbeta, eps, Ceps, nlsf)
spatial_fcn = Krig_Evaluate_Spatial_Function (obs, num_spat_fcns, num_obs_points, nlsf)
trndOBS = matmul(F, beta)
resOBS(:) = obs%field_psqm(:) - trndOBS(:)
OutputUK
stop


NLSF_Least_Sq_Fit(obs, nlsf)
type(Grid_Data_Class), intent(in) :: obs
type(NLSFPar), intent(inout) ::nlsf(*)
Performs a brute force least squares fit of nonlinear spatial function parameters to data points in observation data
nsf > 0
nsf == 0
! initialize residual 0
residual_vect(1:num_obs_points) = obs%field_psqm(1:num_obs_points)
j=1, nsf_local
if(is_reset) residual_vect(:) = obs%field_psqm(1:num_obs_points)
k = nlsf(j)%pre_cond_fcn_num
k .eq.0
TRUE
FALSE
field_precond(:) = NLSF_Eval_Semivariance(obs, nlsf(k))
field_precond(:) = 1.D0
residual_vect = NLSF_Fit_Function(obs, nlsf(j), residual_vect, field_precond)
nlsf(j)%rms = sqrt(sum(residual_vect(:)**2)/float(num_obs_points))
rms(j) = nlsf(j)%rms
residuals(:, j) = residual_vect(:)
is_reset
FALSE
!sort functions into increasing rms
do k = 1,nsf_local; j = minloc(rms(1:nsf_local),1); RankIndx(k) = j; rms(j) = huge(1.D0)
enddo
nlsf(1:nsf_local) = nlsf(RankIndx(1:nsf_local))
residuals(1:num_obs_points, 1:nsf_local) = residuals(1:num_obs_points,RankIndx(1:nsf_local))
nsf_local = min(nsf_local,nsflim_local)
mu = Compute_MEAN(obs%field(1:num_obs_points), num_obs_points)
rms0 = Compute_RMS(obs%field(:) - mu, num_obs_points)
j = 1
do while((nlsf(j)%rms .lt. 0.9_dp * rms0 + 0.1_dp * nlsf(1)%rms) .and. (j+1 .lt. nsf_local)); j = j+1
enddo
j = j – 1; nsflim = j; nsf = j
return


function NLSF_Fit_Function(obs, nlsf, y, f)
function NLSF_Fit_Function(obs, nlsf, y, f)
type(Grid_Data_Class), intent(in) :: obs
type(NLSFPar), intent(inout) ::nlsf
real(dp), intent(in):: y(*), f(*)
real(dp) :: NLSF_Fit_Function(1:obs%num_points)
do j = 1, np; p(j) = float(j-1) / float(np-1); enddo
rms = sqrt(sum(residual(:)**2)/float(num_points))
smoothness_penalty = NLSF_Smooth_Penalty(nlsf)
spf = rms / smoothness_penalty
x0(1:np) = nlsf%f0_min + (nlsf%f0_max - nlsf%f0_min) * p(1:np)
lambda(1:np) = nlsf%lambda_min + (nlsf%lambda_max - nlsf%lambda_min) * p(1:np)
err_min = Huge(1.D0)
x0hat = x0(1)
lambdahat = lambda(1)
j = 1:np
k = 1:np
nlsf%f0 = x0(j)
nlsf%lambda = lambda(k)
s(:) = NLSF_Eval_Semivariance(obs, nlsf)
lpr = LSF_Simple_Linear_Regression(y(:), s(:) * f(:), num_points, alpha, beta)
rms = Compute_RMS((y(1:num_points) - lpr(1:num_points), num_points)
smoothness_penalty = NLSF_Smooth_Penalty(nlsf)
err = rms + spf * smoothness_penalty
err < err_min
T
err_min = err
x0hat = x0(j)
lambdahat = lambda(k)
Compute_RMS(x, n)
return
sqrt(sum(x(1:n)**2) / float(n))
nlsf%f0 = x0hat
nlsf%lambda = lambdahat
s(:) = NLSF_Eval_Semivariance(obs, nlsf)
lpr = LSF_Simple_Linear_Regression(y(:), s(:) * f(:), num_points, alpha, beta)
NLSF_Fit_Function(1:num_points) = y(1:num_points) - lpr(1:num_points)
return
LSF_Simple_Linear_Regression(1:n)
Sxy = sum(x(1:n) * y(1:n))
Sx = sum(x(1:n))
Sy = sum(y(1:n))
Sx2 = sum(x(1:n) * x(1:n))
beta = (n * Sxy - Sx * Sy) /
(n * Sx2 - Sx * Sx)
alpha=(Sy / n) - (beta*Sx / n)
return
alpha + beta * x(1:n)


functionKrig_Eval_Spatial_Function(p,
num_spat_fcns, n_dim, nlsf, save_data)
type(Grid_Data_Class), intent(in) :: p
type(NLSF_Class), intent(in):: nlsf(*)
integer, intent(in) :: n_dim
integer, intent(in) :: num_spat_fcns
Krig_Eval_Spatial_Function(:, 1) = 1.0
j = 1, nsf
return
s(:) = NLSF_Eval_Semivariance(p, nlsf(j))
k = nlsf(j)%precon
k == 0
k==0
k>0
Krig_Eval_Spatial_Function(:, j+1) = s(:)
field_precond(:) = NLSF_Eval_Semivariance(p, nlsf(k))
Krig_Eval_Spatial_Function(:, j+1) = s(:) * fpc(:)


function NLSF_Eval_Semivariance(p, nlsf)
type(Grid_Data_Class), intent(in):: p
type(NLSF_Class), intent(in)::nlsf
nlsf%axis
'x'
'y'
'z'
'x+y'
default
x(:) = p%x(:)
x(:) = p%y(:)
x(:) = p%z(:)
x(:) = p%x(:)+p%y(:)
ERROR MSG
ls_truncate_range
T
F
if(x(j).lt.nlsf%f_range(1)) x(j) = nlsf%f_range(1)
if(x(j).gt.nlsf%f_range(2)) x(j) = nlsf%f_range(2)
nlsf%form
'Logistic'
'Gaussian'
'SinExp'
'CosExp'
default
NLSF_Eval_Semivariance(:) = 1 /
(1 + exp(- (x(:s) - nlsf%f0) / nlsf%lambda))
NLSF_Eval_Semivariance(:) =
exp(- ((x(:) - nlsf%f0) / nlsf%lambda)**2)
NLSF_Eval_Semivariance(:) =
sin((x(:)-nlsf%f0)/nlsf%lambda)
* exp(-((x(:) - nlsf%f0)/nlsf%lambda)**2)
NLSF_Eval_Semivariance(:) =
cos((x(:)-nlsf%f0)/nlsf%lambda)
* exp(-((x(:) - nlsf%f0)/nlsf%lambda)**2)
ERROR MSG
return

Krig_Generalized_Least_Sq

Compute the Univeral Kriging linear estimate of f following Cressie 1993 pages 151-154

Krig_Compute_Distance
(obs, obs, distance_horiz, distance_vert, num_obs_points)

Gamma = Krig_Compute_Variogram
(num_obs_points, num_obs_points, distance_horiz, distance_vert, num_obs_points, par)

Fs = Krig_Eval_Spatial_Function
(obs, num_spat_fcns, num_obs_points)

Krig_Compute_Distance
(obs, grid, D0h, D0z, num_obs_points)

gamma0 = Krig_Compute_Variogram
(num_obs_points, num_points, D0h, D0z, num_obs_points, par)

Fs0 = Krig_Eval_Spatial_Function
(grid, num_spat_fcns, num_points, nlsf, save_data)

$$\Gamma_U \equiv \begin{cases} \gamma(\mathbf{s}_i - \mathbf{s}_j), & i = 1, \ldots, n, j = 1, \ldots, n, \\ f_{j-1-n}(\mathbf{s}_i), & i = 1, \ldots, n, j = n+1, \ldots, n+p+1, \\ 0, & i = n+1, \ldots, n+p+1, \\ & j = n+1, \ldots, n+p+1, \end{cases}$$

n+p+1?

```
n = num_obs_points      p = num_spat_fcns
Fs0T=transpose(Fs0)
FsT=transpose(Fs)
R(1:n+p,   1:n+p)  = 0.
R(1:n,     1:n)    = gamma(1:n, 1:n)
R(1:n,     n+1:n+p)= Fs(1:n, 1:p)
R(n+1:n+p, 1:n)    = FsT(1:p, 1:n)

R(n+1:n+p, n+1:n+p)= 0.
```