Implementation and Application of the Kalman Filter Data Assimilation Approaches in NASA's Land Information System Infrastructure

X. Zhan[1], S.V. Kumar[2], W. Crow[1], K. Arsenault[2], P. Houser[3], C. Peters-Lidard[2]

[1] USDA-ARS Hydrology and Remote Sensing Lab, Beltsville, MD
[2] NASA-GSFC Hydrological Sciences Branch, Greenbelt, MD
[3] George Manson University & Center for Research on Environment and Water, Calverton, MD

Motivation

Many land surface **satellite data products** have been or will be available from NASA, NOAA, ESA, & JAXA satellites (e.g., NDVI/EVI, LAI, LST, Snow, & SM from MODIS, AMSR etc);

Many land surface data assimilation algorithms such as the **Kalman filters** have been published in the literature;

Based the successful development of NLDAS & GLDAS, the **Land Information System (LIS)** could become the best test bed for applying the DA algorithms to assimilate the available satellite data products into land surface models for operational numerical weather predictions in addition to its other applications;

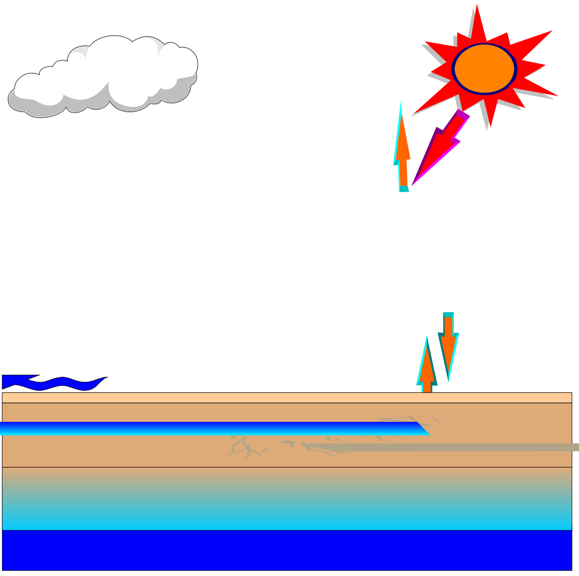
Therefore, **implementing and evaluating** the Kalman filter DA assimilation algorithms in LIS be

Topography,
Soils

Land Cover,
Vegetation
Properties

Meteorology

Snow
Soil Moisture
Temperature



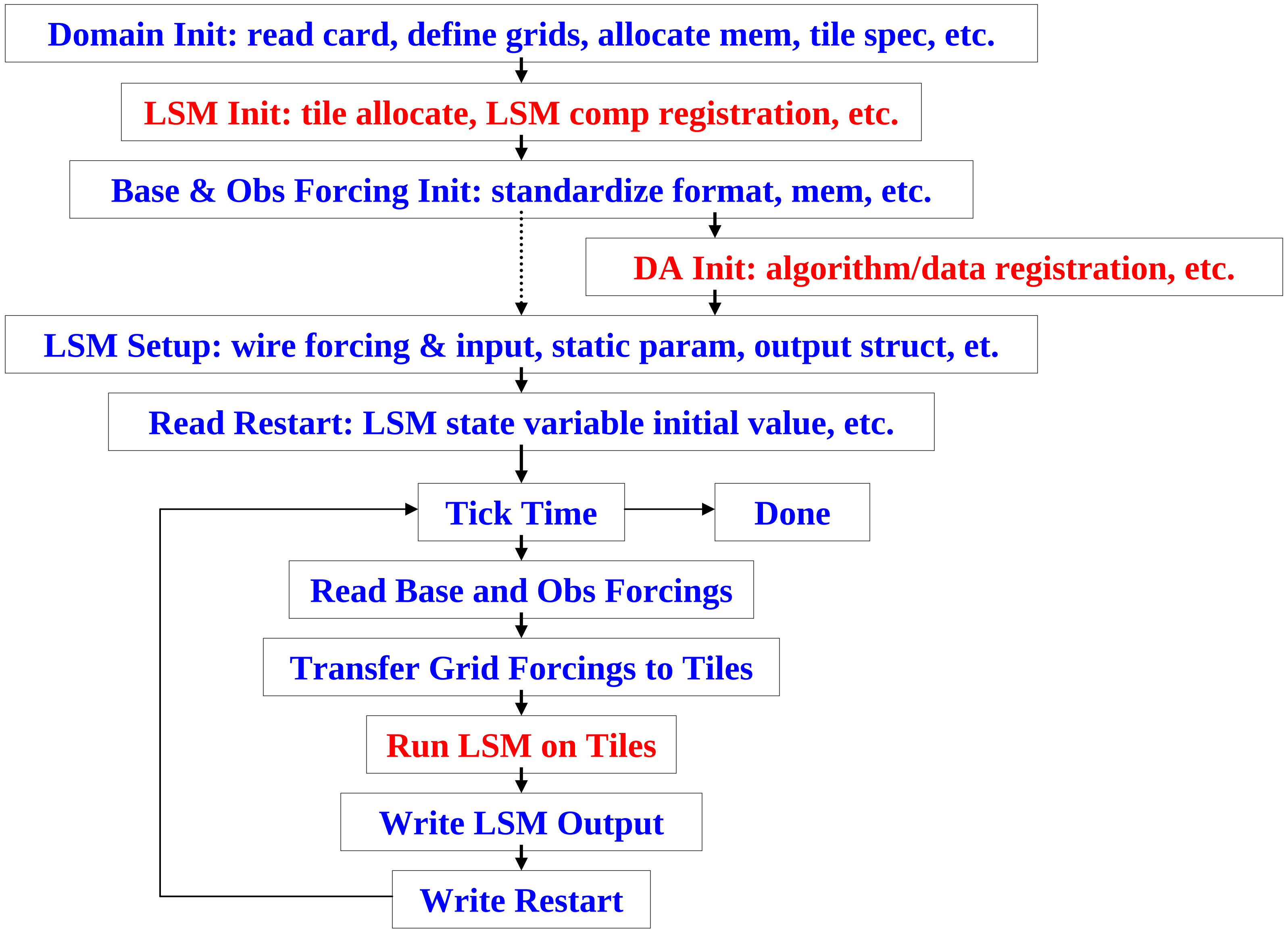
Land Information System (LIS)

LIS is a NASA “software of year” **award winning software** system to enhance the computational capability of the Land Data Assimilation Systems (NLDAS and GLDAS);

LIS coding is based on the Fortran subroutines and functions of the original LDAS Systems, but is restructured using new technologies such as **object-oriented programming**, parallel and low cost cluster computing so that LIS can run land surface models for any size of domain (global or any region), any spatial resolution (0.25 degree, 1km, 30m or any other);

The O-O programming technology allows **adding new feathers** (such as **data assimilation**) to LIS as new plug-ins without major modifications to the original LIS structure and code.

LIS Flow Chart and Data Assimilation Implementation



Domain Init: read card, define grids, allocate mem, tile spec, etc.
LSM Init: tile allocate, LSM comp registration, etc.
Base & Obs Forcing Init: standardize format, mem, etc.
DA Init: algorithm/data registration, etc.
LSM Setup: wire forcing & input, static param, output struct, et.
Read Restart: LSM state variable initial value, etc.
Tick Time
Done
Read Base and Obs Forcings
Transfer Grid Forcings to Tiles
Run LSM on Tiles
Write LSM Output
Write Restart

```
LIS main core/lisdrv.F90:

  call LIS_config
  call LIS_domain_init
  call LIS_lsm_init
  call LIS_baseforcing_init
  call LIS_obsforcing_init
  call LIS_dataassim_init
  call LIS_setuplsm
  call LIS_readrestart
  do while (.NOT. LIS_endofrun())
     call LIS_ticktime
     call LIS_setDynlsm
     call LIS_get_base_forcing
     call LIS_get_obs_forcing
     call LIS_force2tile
     call LIS_lsm_main
     call LIS_lsm_output
     call LIS_writerestart
  enddo
```

```
LIS_lsm_init in plugins/lsm_pluginMod.F90:

    ...
  call registergetstatevar(1,1,noah_getSMStVar)
  call registersetstatevar(1,1,noah_setSMStVar)
  call registerobstransform(1,1,noah_obs2st_vsm)
  call registerlsmreset(1,noah_reset)
  call registerlsmpertbf(1,noah_pertbf)
  call registerlsmpertbs(1,noah_pertbs)
  call registerqa_check(1,1,noah_qa_sm)
    ...
  call registergetstatevar(4,1,mos_getSMStVar)
  call registersetstatevar(4,1,mos_setSMStVar)
  call registerobstransform(4,1,mos_obs2st_vsm)
  call registerreset(4,mos_reset)
  call registerlsmpertbf(4,mos_pertbf)
  call registerlsmpertbs(4,mos_pertbs)
  call registerlsmqa_check(4,1,mos_qa_sm)
    ...
```

These calls are to register the subroutines for the specific LSM and observed state variable. The code for these subroutines are deposited in the subdirectory corresponding to the LSM. For example, the subroutine `noah_pertbf` is `noah_pertbf.F90` in `lsms/noah.2.6`.

The called registerxxx functions are defined in `core/LIS_lsm_FTable.c`

LIS-DA Code for LSM-Specific Subroutine Registration

```
LIS main core/lisdrv.F90:

  call LIS_config
  call LIS_domain_init
  call LIS_lsm_init
  call LIS_baseforcing_init
  call LIS_obsforcing_init
  call LIS_dataassim_init
  call LIS_setuplsm
  call LIS_readrestart
  do while (.NOT. LIS_endofrun())
     call LIS_ticktime
     call LIS_setDynlsm
     call LIS_get_base_forcing
     call LIS_get_obs_forcing
     call LIS_force2tile
     call LIS_lsm_main
     call LIS_lsm_output
     call LIS_writerestart
  enddo
```

```
LIS_dataassim_init in core/dataassim_module.F90:

  call dataassim_plugin
  call daobs_plugin
  call dasetup(lis%a%daalg,lis%a%davar)
  call daobssetup(lis%a%daobs)


DA plugin in plugins/dataassim_pluginMod.F90:
   call registerdasetup(1,1,sm_di_init)
   call registerfrcst(1,1,sm_di_frcst)
   call registerupdate(1,1,sm_di_update)
   call registerdasetup(2,1,sm_ekf_init)
   call registerfrcst(2,1,sm_ekf_frcst)
   call registerupdate(2,1,sm_ekf_update)
   call registerdasetup(3,1,sm_enkf_init)
   call registerfrcst(3,1,sm_enkf_frcst)
   call registerupdate(3,1,sm_enkf_update)

OBS plugin in plugins/dataobs_pluginMod.F90:
    call registerdaobssetup(1,smobs_setup)
    call registerdaobsread(1,read_tmism)
    call registerdaobssetup(2,smobs_setup)
    call registerdaobsread(2,read_synsm)
```

The method-specific subs such as `sm_enkf_assim` is in `dataassim/enkf/sm/smenkfdrv_module.F90`. The `dataassim_plugin` and `daobs_plugin` subs are to register the DA and OBS subs. The called registerxxx functions are in `core/LIS_dataassim_FTable.c`

LIS-DA Code for DA Method-Specific Subroutine Registration

LIS-DA Code for DA Execution Procedure at Each Time Step

```
LIS main core/lisdrv.F90:

  call LIS_config
  call LIS_domain_init
  call LIS_lsm_init
  call LIS_baseforcing_init
  call LIS_obsforcing_init
  call LIS_dataassim_init
  call LIS_setuplsm
  call LIS_readrestart
  do while (.NOT. LIS_endofrun())
     call LIS_ticktime
     call LIS_setDynlsm
     call LIS_get_base_forcing
     call LIS_get_obs_forcing
     call LIS_force2tile
     call LIS_lsm_main
     call LIS_lsm_output
     call LIS_writerestart
  enddo
```

```
LIS_lsm_main in core/lsm_module.F90:

    if(lis%a%daalg.gt.0) then
       call daobsread(lis%a%daobs)
       call forecast(lis%a%daalg,lis%a%davar)
       call update(lis%a%daalg,lis%a%davar)
    else
       call lsmrun(lis%d%lsm)
    endif
```

The variables are read from namelists in lis.crd file:

```
&data_assimilation
LIS%a%DAALG = 3  ! 0-no,1-di,2-ekf,3-enkf da mehod
LIS%a%DAOBS = 2  ! 0-no,1-tmi_sm,2-syn_sm,3-amsre_sm, …
LIS%a%DAVAR = 1  ! 0-no,1-sm,2-ts,
LIS%a%NSTV  = 16 ! # of LSM state variables
LIS%a%NDAV  = 4  ! # of prog model vars to be assimilated
LIS%a%NENS  = 5  ! number of ensemble members for EnKF
LIS%a%RENSEM= 0  ! write ensember data into files (0-n,1-y)
LIS%a%fvlfn = "./data/perturb/f_mos.dat"  ! forc perturb’ns
LIS%a%svlfn = "./data/perturb/s_mos.dat"  ! stat perturb’ns
/

&soil_moisture_da
smobsdir  = "./data/NOAHSoilMoist1_NLDAS" ! data location
nob       = 1      ! # of obs vars (e.g. surface sm)
oer       = 0.01   ! obs error rate
Mer       = 0.03   ! Mdl error rate
/
```

Data Assimilation Algorithms Implemented

Direct Insertion (DI): **replace LSM states when corresponding observation data are available:**

$$X_a = Z$$

Extended Kalman Filter (EKF): **correct LSM states by weighing model forecasts and observations with their error covariance:**

$$K = P{*}H^T/[H{*}P{*}H^T + R]$$

$$P = (I\text{-}K{*}H)P$$

$$X_a = X_b + K{*}[Z - h(X_b)]$$

Ensemble Kalman Filter (EKF): **correct LSM states by weighing model forecasts and observations with their error covariance:**

$$P_{ym} = \Sigma(X_y\text{-}\mu_y)(X_m\text{-}\mu_m)/n$$

$$K = P_{ym}/(P_{mm} + R)$$

$$X_a = \Sigma[X_b + K{*}(Z - X_b)]/n$$

Run Data Assimilation In LIS

```
&data_assimilation
LIS%a%DAALG = 3  ! 0-no,1-di,2-ekf,3-enkf da mehod
LIS%a%DAOBS = 2  ! 0-no,1-tmi_sm,2-syn_sm,3-amsre_sm, …
LIS%a%DAVAR = 1  ! 0-no,1-sm,2-ts,
LIS%a%NSTV  = 16 ! # of LSM state variables
LIS%a%NDAV  = 4  ! # of prog model vars to be assimilated
LIS%a%NENS  = 5  ! number of ensemble members for EnKF
LIS%a%RENSEM= 0  ! write ensember data into files (0-n,1-y)
LIS%a%fvlfn = "./data/perturb/f_mos.dat"  ! forc perturb’ns
LIS%a%svlfn = "./data/perturb/s_mos.dat"  ! stat perturb’ns
/

&soil_moisture_da
smobsdir   = "./data/NOAHSoilMoist1_NLDAS" ! data location
nob        = 1      ! # of obs vars (e.g. surface sm)
oer        = 0.01   ! obs error rate
Mer        = 0.03   ! Mdl error rate
/
```

Data Sets for Testing the DA Features in LIS

1. **SGP'99 TMI observations:** Resampled and calibrated soil moisture retrievals from TRMM Microwave Imager (TMI) for the SGP'99 domain for the days from 8 to 20th of July, 1999.

2. **Synthetic 0-2cm SM from Mosaic or Noah LSMs:** simulated soil moisture Mosaic and Noah LSM in LIS and the NLDAS atmospheric forcing data set for the days from 18 June to 20 July, 2002;

3. **NASA's AMSR-E soil moisture data product:** Level 3 soil moisture retrievals since 18 June, 2002

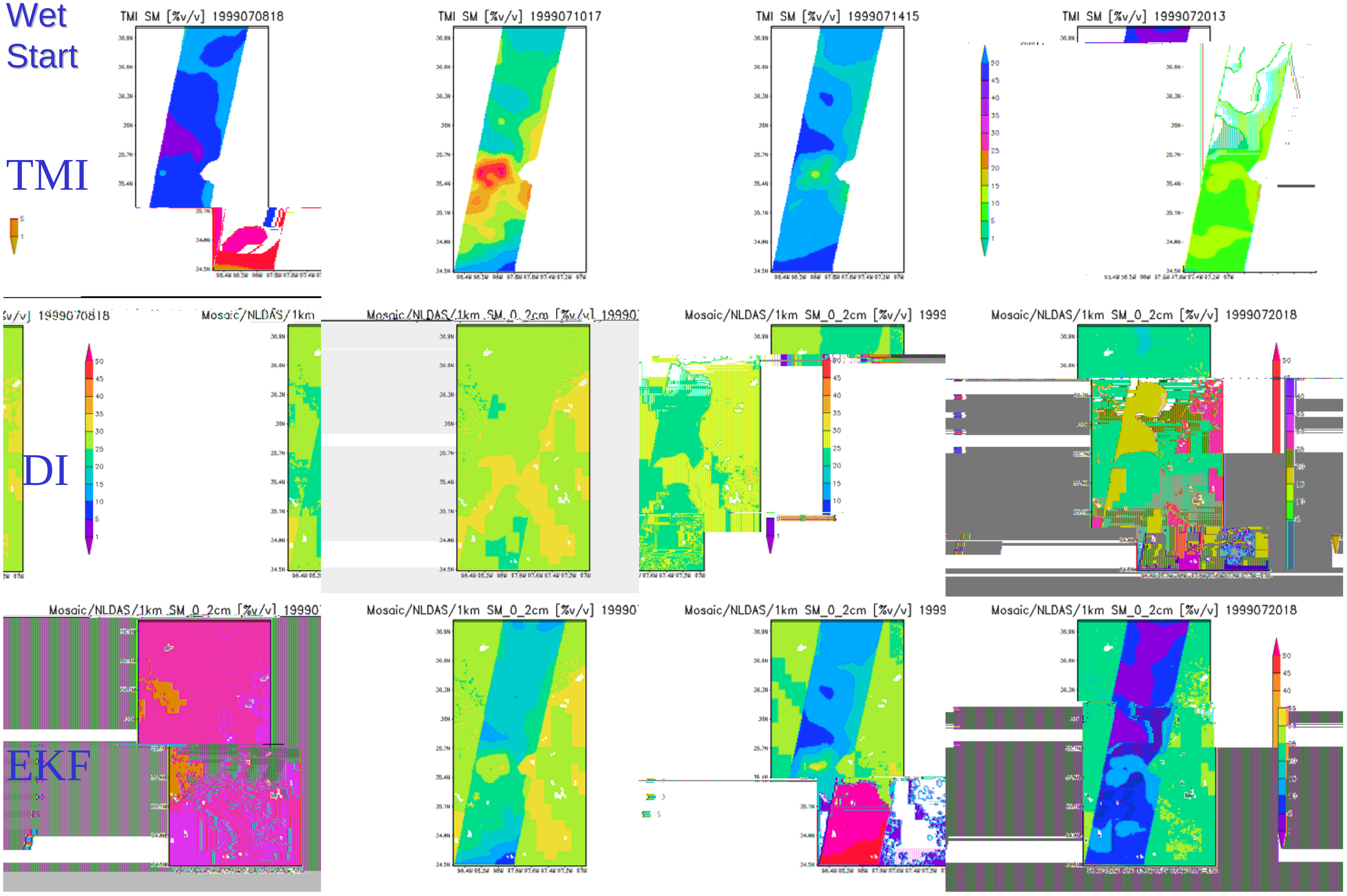
SGP'99 TMI SM Data Assimilation with Mosaic LSM

Wet Start

TMI

DI

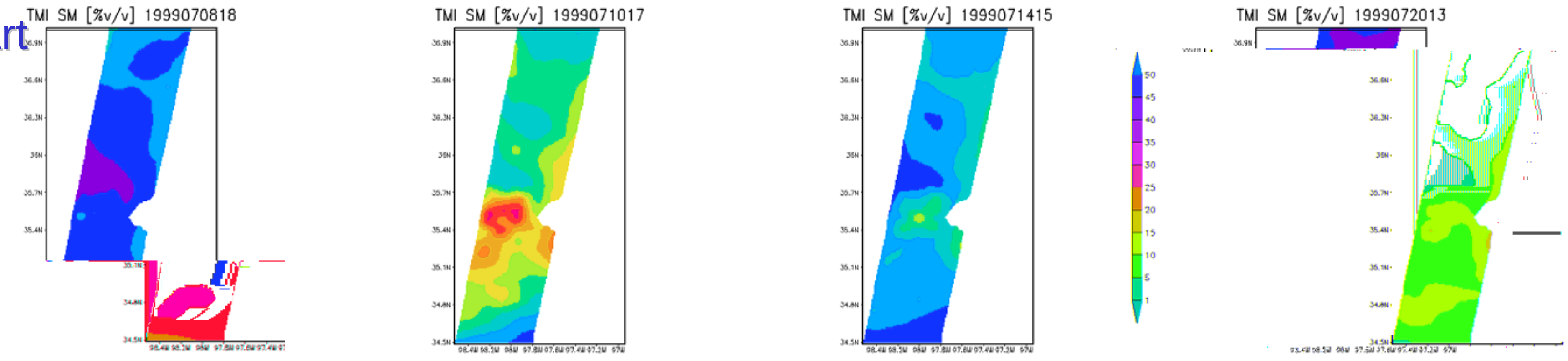
EKF



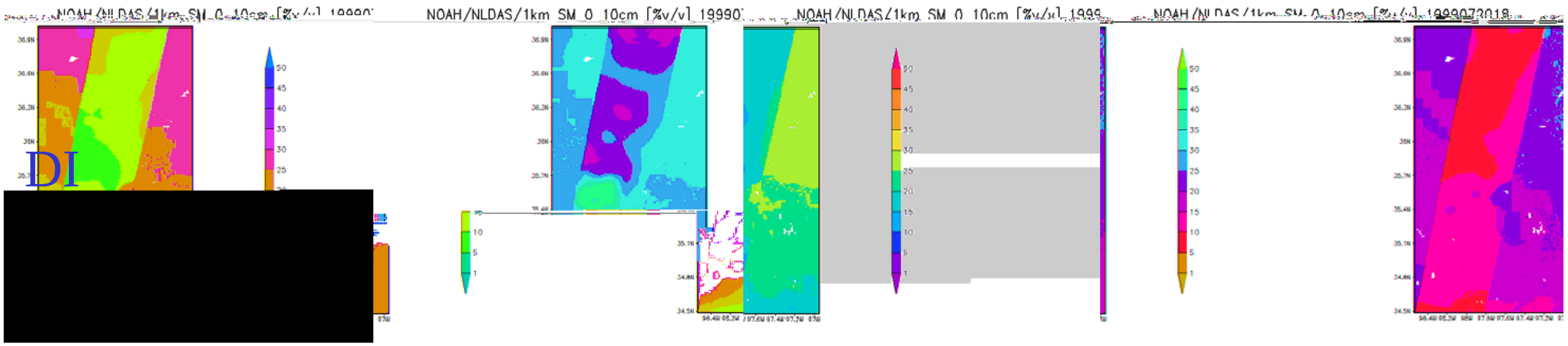
SGP'99 TMI SM Data Assimilation with Noah LSM

ReStart

TMI

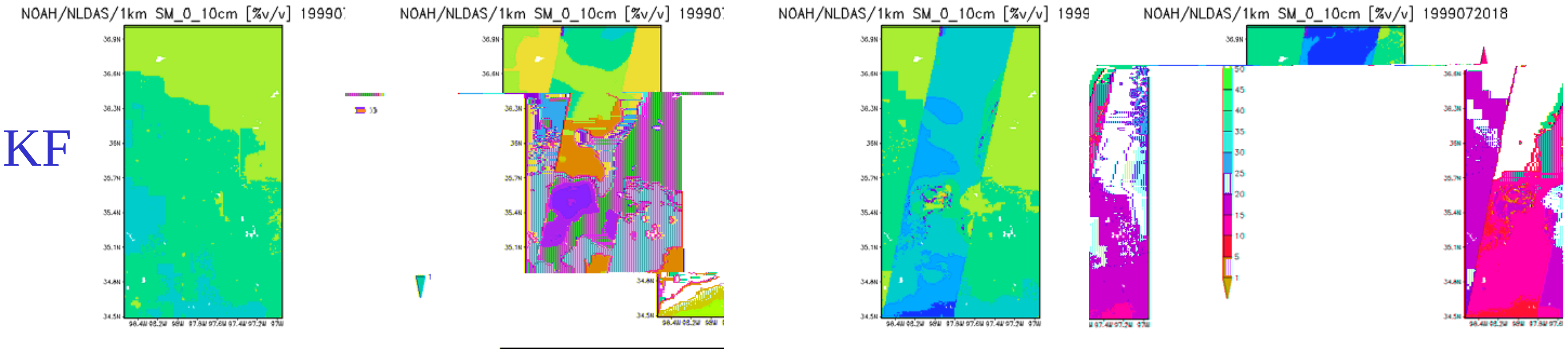


DI



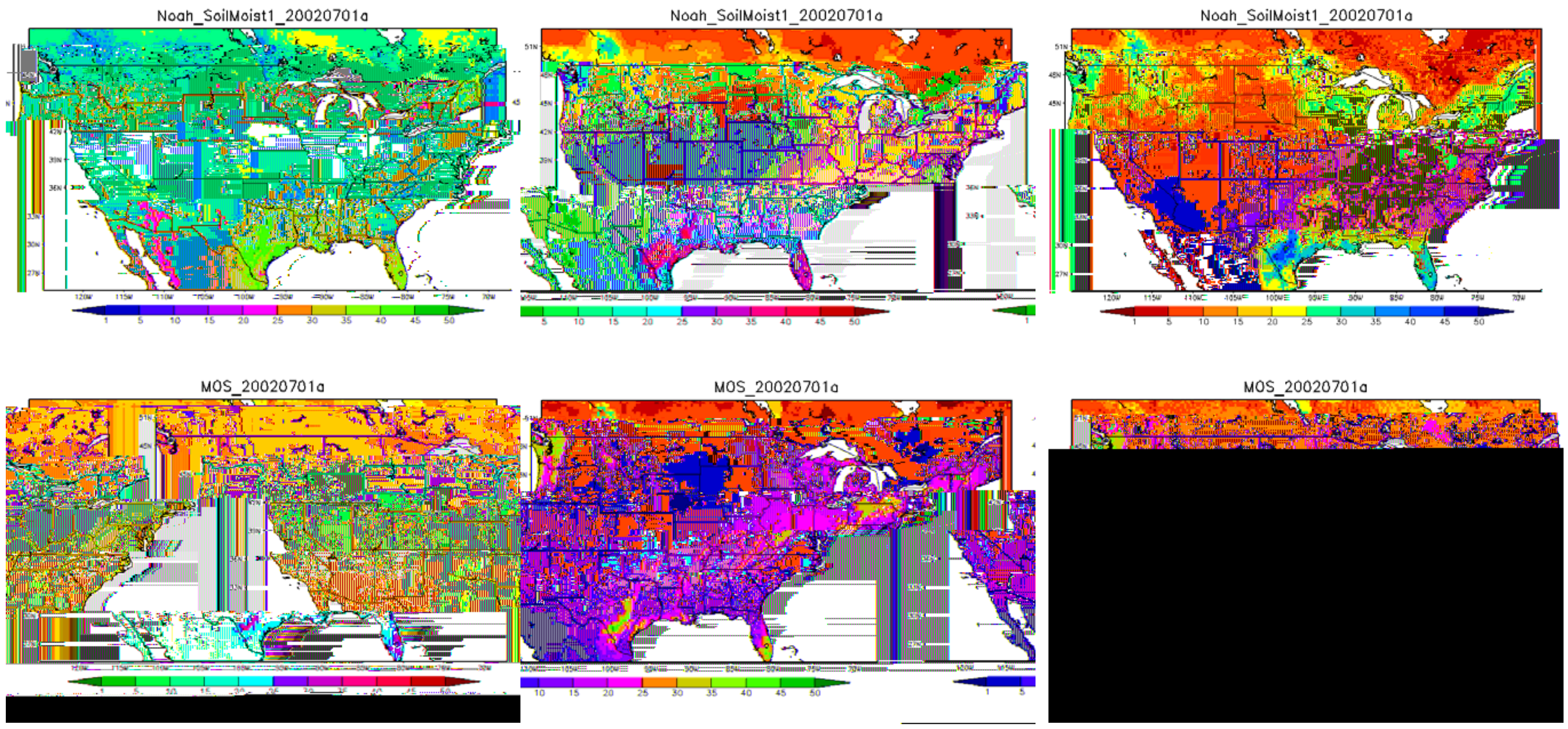
NOAH/NLDAS/1km SM_0_10cm [%v/v] 19990
NOAH/NLDAS/1km SM_0_10cm [%v/v] 19990
NOAH/NLDAS/1km SM_0_10cm [%v/v] 1999
NOAH/NLDAS/1km SM_0_10cm [%v/v] 1999072018

EKF



NOAH/NLDAS/1km SM_0_10cm [%v/v] 19990
NOAH/NLDAS/1km SM_0_10cm [%v/v] 19990
NOAH/NLDAS/1km SM_0_10cm [%v/v] 1999
NOAH/NLDAS/1km SM_0_10cm [%v/v] 1999072018

EnKF Assimilation of Synthetic SM Data


Noah_SoilMoist1_20020701a
Noah_SoilMoist1_20020701a
Noah_SoilMoist1_20020701a
MOS_20020701a
MOS_20020701a
MOS_20020701a

Noah

Mosaic

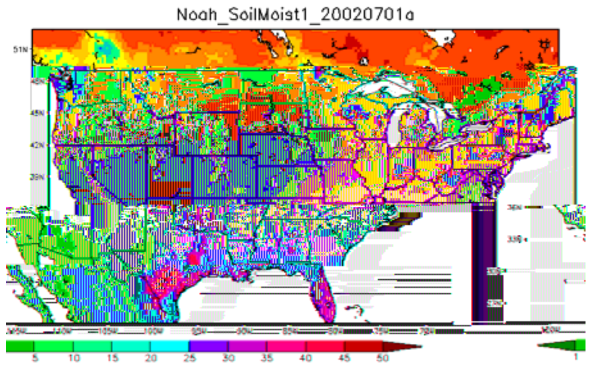
LSM Run

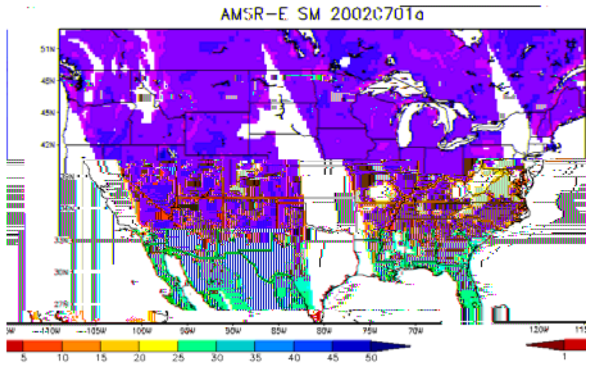
Synthetic Obs

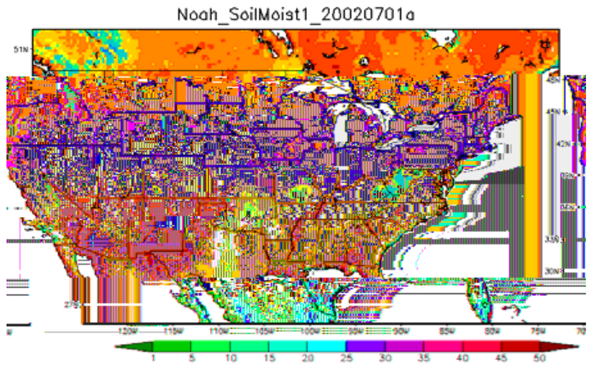
EnKF Assimilation

EnKF Assimilation of AMSR-E SM Retrievals

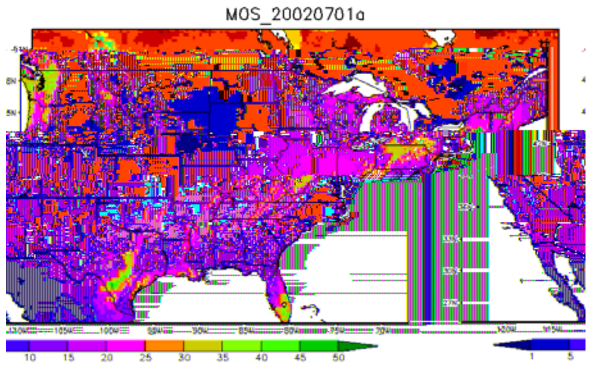
Noah

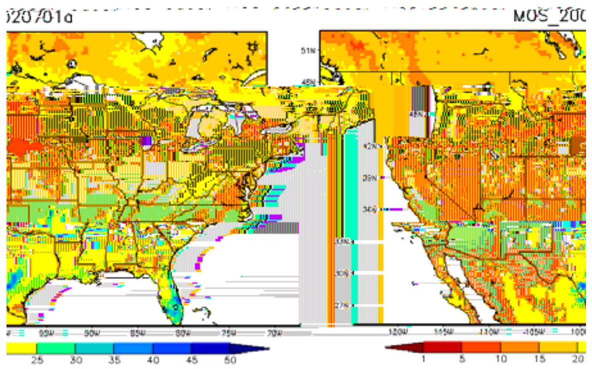






Mosaic





LSM Run

AMSR-E

EnKF Assimilation

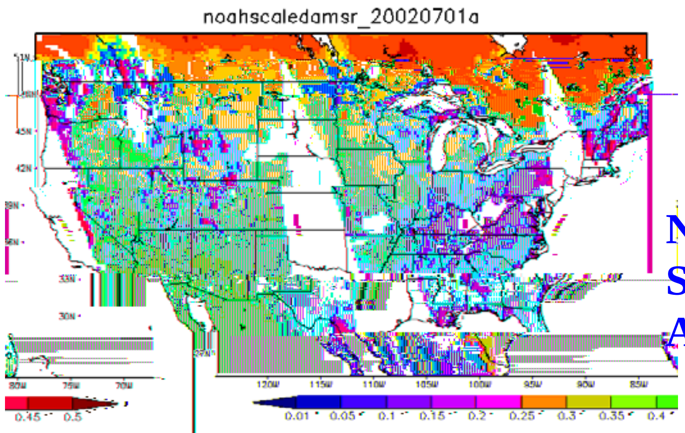
CDF Matching of AMSR-E Retrievals to Model Simulations

Purpose: LSM simulations can have full time and space coverage, but accuracy is uncertain; AMSR-E retrievals are patchy, biased and variations damped. Data assimilation can combine them, but their scales need to be matched for DA methods such as EnKF;

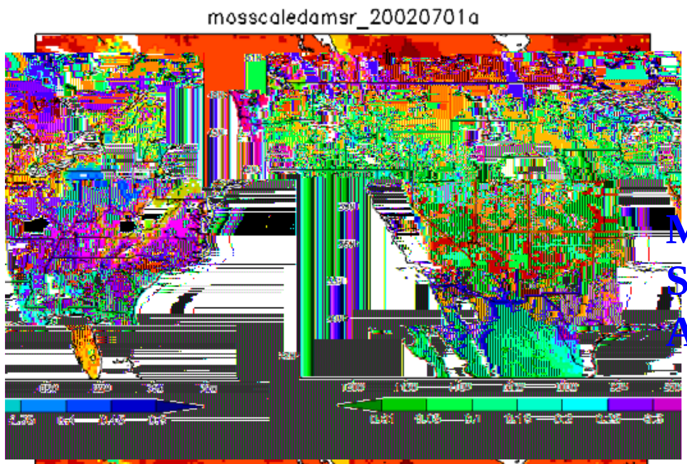
The cumulative distribution function matching method used in Reichle & Koster (2004) can scale the AMSR-E retrievals to the scales of the simulations of LSMs to be used for assimilation;

How will the scaled AMSR-E data be compared with the model simulations?

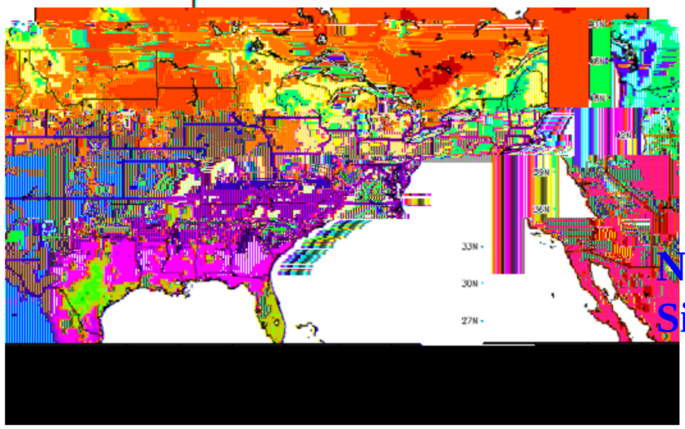
Effect of Scaling AMSR-E to Model Simulations



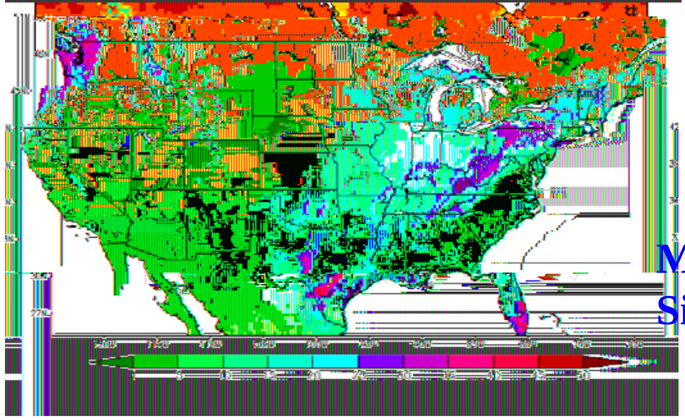
Noah-Scaled AMSR-E



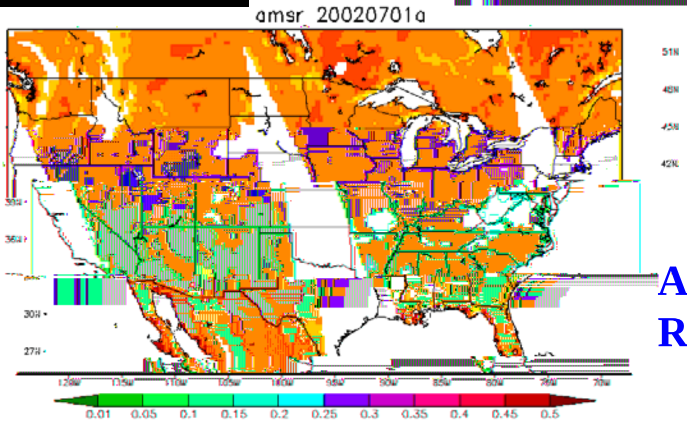
Mosaic-Scaled AMSR-E



Noah Simulation

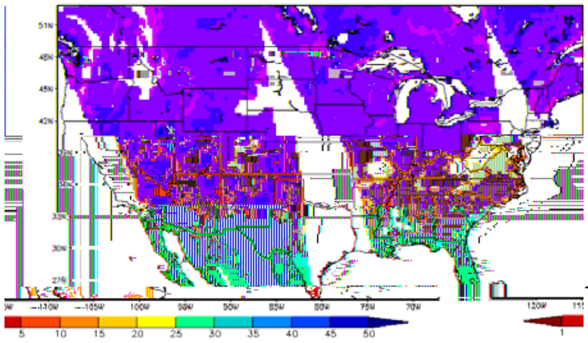


Mosaic Simulation



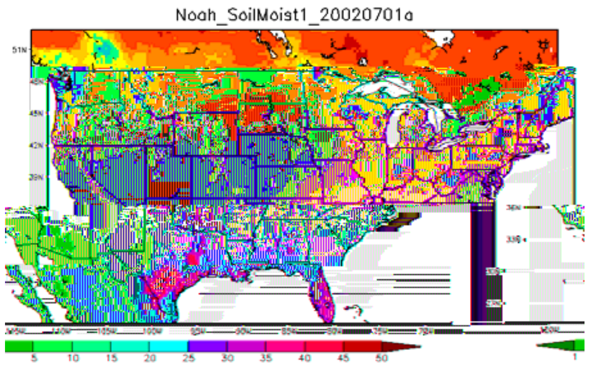
AMSR-E Retrieval

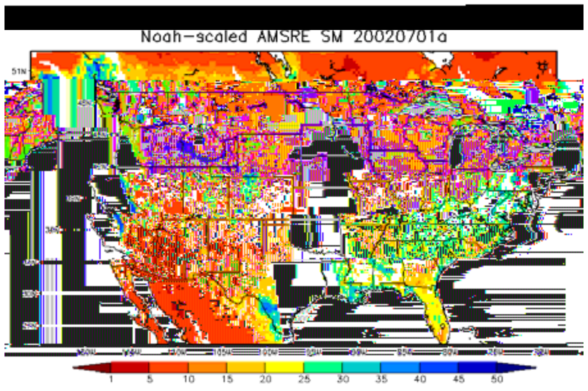
EnKF Assimilation of Scaled AMSR-E SM Retrievals

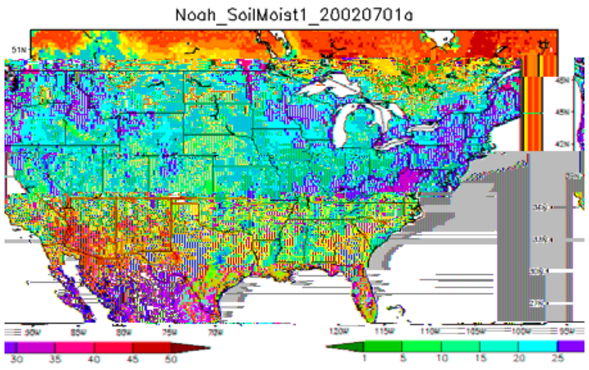


AMSR-E SM

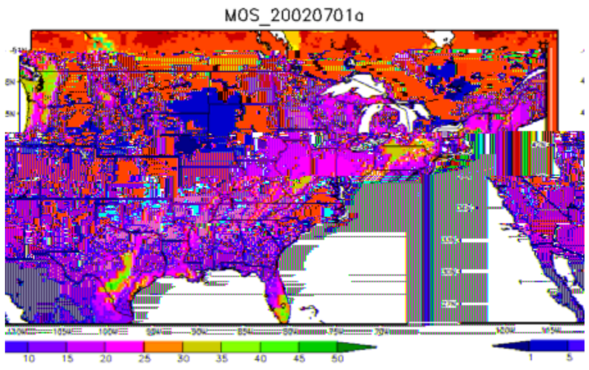
Noah

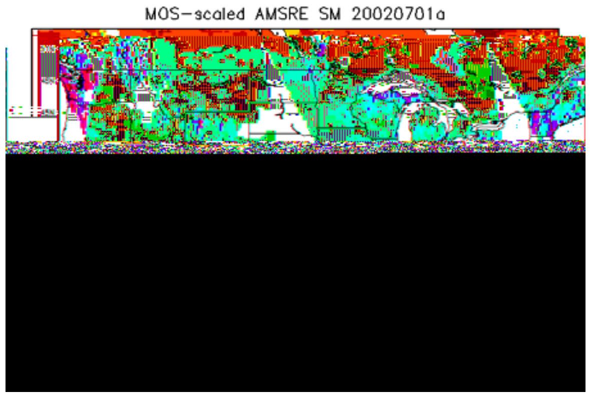


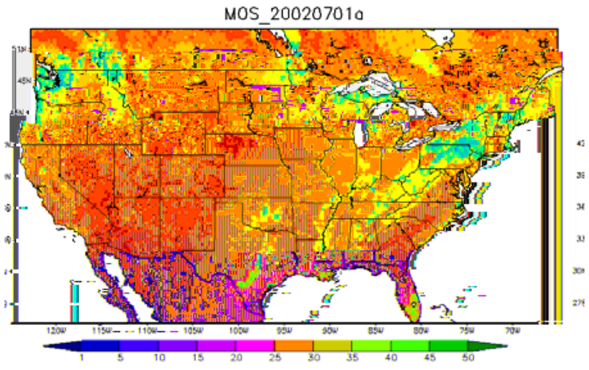




Mosaic







LSM Run

Scaled AMSR-E

EnKF Assimilation

Ensemble Generation & Model Error Covariance

Model error covariance depends on the magnitude of perturbations added to the ensemble generation. Therefore, the EnKF assimilation result is subject to the perturbation setup;

Dee (1995) and Crow et al. (2005): Standardized innovation

$$[Z-X_b]^T[Z-X_b]/[P_{mm}+R] \sim N(\mu,1)$$

for the assimilated states to be optimal.

A post processing tool to compute the standardized innovation variance is created for adjusting the perturbation setup manually.

Summary

Three data assimilation algorithms (DI, EKF, EnKF) have been implemented in the Land Information System (LIS) to assimilate soil moisture observations into Mosaic or Noah LSM simulations;

The LIS DA capability has been tested with difference soil moisture observation data sets;

A tool for generating the ensembles of EnKF is created to ensure the assimilated result is optimal;

Further quantitative evaluation is needed and will be made before we can employ the LIS DA function for large scale applications.