OOPS-3DVAR workflow

Understanding OOPS-3DVAR C++ workflow using DDT. HARMONIE CY46

Pau Escribà and Oriol Farrés
(with the help of Roel Stappers)

Summer 2023

OUTLINE

- Purpose
- Setup of the presentation
- 3DVAR OOPS CY46 workflow

Purpose

- The objetive of this work is to describe a real 3DVAR OOPS experiment workflow, in CY46.

- This work aims to be a reference to understand de C++ OOPS code and to be used in further developments (4DVAR OOPS LELAM)

Setup of the presentation

- Each workflow step will be given a hierarchical numbering from the beginning to the end
- For example, 1.3.4 means that we are in step 4 of step-father 3 of step-grandfather 1
- A brief description of each step will be given. Code screenshots can also be attached.

3DVAR OOPS CY46 workflow (attached PDF file)

0. *srun ifs4dvar oops.json*

- Launch command at script level
- ifs4dvar: C++ compiled binary
- oops.json: json file with OOPS experiment config info

1. ***Ifs4dvar.cc: main.*** Main C++ routine

```
26  int main(int argc, char ** argv) {
27    ifs::mpi_init_f90();
28
29    ifs::RunIFS run(argc, argv);
30
31    ifs::instantiateObsErrorFactory();
32    ifs::instantiateTlmFactory();
33    ifs::instantiateIFSMatricesFactory();
34
35    oops::Variational<ifs::IfsTraits> var;
36
37    int nt = omp_get_max_threads();
38    Log::info() << "Maximum number of OpenMP threads: " << nt << std::endl;
39
40    int test=0;
41    while ((::getenv("OOPS_TEST_DEBUG")!=0) && (test==0)) {
42      sleep(1);
43    }
44    run.execute(var);
45    return 0;
46  };
```

1.1 mpi_init_f90

```
25
26  int main(int argc, char ** argv) {
27    ifs::mpi_init_f90();
28
```

- **Initialize MPI and OpenMP**: mpi_init_f90 (C++) eventually calls **mpI_init.F90**

- **ifs**::mpi_init_f90(): **ifs** is a *namespace* of C++. It is equivalent to a *library***:** defined block that contains a set of routines/elements

1.2 *RunIFS.cc: RunIFS(argc, argv)*

```
28
29      ifs::RunIFS run(argc, argv);
30
```

- **argc**: integer, number of arguments

- **argv**: pointer to pointer to array of chars, where each array element is a (string) argument of the ifs4dvar binary at script level. The last element is the name of the json config file, “oops.json”

- **Instantation of object run** of class **RunIFS. argc and argv** are passed as arguments to the *constructor of the class.*

1.2 *RunIFS.cc: RunIFS(argc, argv)*

```
18
19 RunIFS::RunIFS(int argc, char ** argv) : oops::Run(argc, argv) {
20   Log::trace() << "Creating RunIFS" << std::endl;
21   set_err_trap_();
22   const eckit::Configuration * confptr = &config();
```

- **L19**: **oops::Run(argc, argv),** constructor list. Besides, class Run from namespace oops is **inherited** by RunIFS. It means all methods and members of Run can be used from RunIFS.

- **Jump to 1.2.1 *Run.cc: Run(argc, argv)***

1.2.1 *Run.cc: Run(argc, argv)*

```
30
31  Run::Run(int argc, char** argv) : eckit::Main(argc, argv, "OOPS_HOME"), config_(), timer_() {
32  // Get configuration file from command line
33    ASSERT(argc >= 2);
34    eckit::PathName configfile = argv[argc - 1];
35
36  // Read configuration
37    config_.reset(new eckit::JSONConfiguration(configfile));
```

- **L31**: construct class Main from **eckit** ECMWF C++ external library to be able to read json config file

- **L37.** Read json config file and store the content in object **config_**

1.2 *RunIFS.cc: RunIFS(argc, argv)*

```
24  //  ************************************
25  //  *** ifs_init_f90 should disappear ***
26  //  ************************************
27    Log::trace() << "Calling IFS_INIT" << std::endl;
28    mpi_init_f90();
29    ifs_init_f90(&confptr);
30    Log::trace() << "IFS_INIT done" << std::endl;
31  //  ************************************
```

- **L28**: **jump to 1.2.2 *mpi_init_f90.*** Initialize MPI environament. Why here again? (see jump 1.1)

- **L29: jump to 1.2.3 *ifs_init_f90.*** It calls **IFS_INIT.F90.** This routine is important because **it calls a list of SU*.F90 IFS setup routines**, **so part of fort.4 is read here**. This namelist info will define the OOPS experiment!!!

1.3 *instantiateObsErrorFactory.h: instantiateObsErrorFactory*

```
31    ifs::instantiateObsErrorFactory();
32    ifs::instantiateTlmFactory();
33    ifs::instantiateIFSMatricesFactory();
```

- This three calls works equally. The idea is to ***initialize the makers*** of each **Factory**. These makers are classes from where objects will be instantiated further in the code. Only line 31 is described.

- In this step the list of possible makers is defined.
- **Jump 1.3 *instantiateObsErrorFactory*.h: *instantiateObsErrorFactory***

1.3 *instantiateObsErrorFactory*.h: *instantiateObsErrorFactory*

```
13 void instantiateObsErrorFactory() {
14   static oops::ObsErrorMaker<IFS, AllObsCovariance> makerObsErrIFS_("AllObsError");
15 }
```

- Here the concept of C++ *template variable* is used. **IFS** and **AllObsCovariance** are values for the template variables **MODEL** and **T**, in next slide. The compiler substitutes these values like a pre-processor to point to the corresponding class in runtime.

- All template variables are defined **at the beginng of the C++ code**.

1.3 *instantiateObsErrorFactory*.h: *instantiateObsErrorFactory*

```
13 void instantiateObsErrorFactory() {
14   static oops::ObsErrorMaker<IFS, AllObsCovariance> makerObsErrIFS_("AllObsError");
15 }
```

- If a **new maker** wants to be introduced, *only* a line below 14 must be introduced. Also coding thr class for the new maker…
- When constructing object, string “**AllObsError**” is passed as argument
- **Jump to 1.3.1 *ObsErrorBase.h: ObsErrorMaker***

1.3.1 *ObsErrorBase.h: ObsErrorMaker*

```
90  template<class MODEL, class T>
91  class ObsErrorMaker : public ObsErrorFactory<MODEL> {
92    typedef ObservationSpace<MODEL> ObsSpace_;
93    virtual ObsErrorBase<MODEL> * make(const ObsSpace_ & obs, const eckit::Configuration & conf)
94      { return new T(obs.observationspace(), conf); }
95   public:
96    explicit ObsErrorMaker(const std::string & name) : ObsErrorFactory<MODEL>(name) {}
97  };
98
```

- “name” to **jump 1.3.1.1 *ObsErrorBase.h: ObsErrorFactory***

```
101 template <typename MODEL>
102 ObsErrorFactory<MODEL>::ObsErrorFactory(const std::string & name) {
103   if (getMakers().find(name) != getMakers().end()) {
104     Log::error() << name << " already registered in observation error factory." << std::endl;
105     ABORT("Element already registered in ObsErrorFactory.");
106   }
107   getMakers()[name] = this;
108 }
```

- **getMakers** is a map (like a dictionary in Python) that stores all posible makers. This will be used further in the code comparing with the value given in the **json file**

1.4 *instantiateTlmFactory.h: instantiateTlmFactory*

```
31    ifs::instantiateObsErrorFactory();
32    ifs::instantiateTlmFactory();
33    ifs::instantiateIFSMatricesFactory();
```

- L32 jumps:
- **1.4 *instantiateTlmFactory*.h: *instantiateTlmFactory***
- **1.4.1 *LinearModelBase.h: LinearModelMaker***
- **1.4.1.1 *LinearModelBase.h: LinearModelFactory***
- L33 jumps:
- **1.5 *instantiateIFSMatricesFactory*.h: *instantiateIFSMatricesFactory***
- **1.5.1 *GenericMatrixMaker.h: GenericMatrixMaker***
- **1.5.1.1 *GenericMatrixMaker.h: GenericMatrixFactory***

1.6 *Variational.h: Variational*

```
35    oops::Variational<ifs::IfsTraits> var;
36
```

- Here is defined the *template variable* **MODEL=ifs::IfsTraits.** Looking at ***IfsTraits.h*** can be seen that **IFS=ifs::IfsTraits**

- Here also is **instantiated** (declared and constructed) the **superobject var,** which has geometry, model, state and others... See class Variational in *Variational.h.*

- When the object (var) is instantiated without any argument, the ***default constructor*** is called.

1.6 *Variational.h: Variational*

```
46  public:
47  // ----------------------------------------
48    Variational() {
49      instantiateCostFactory<MODEL>();
50      instantiateCovarFactory<MODEL>();
51      instantiateMinFactory<MODEL>();
52      instantiateObsErrorFactory<MODEL>();
53      instantiateTlmFactory<MODEL>();
54    }
55  // ----------------------------------------
```

- L49 to L53, declaration of many ***template variables*** for several **Factories.**

- The working of these callings is equivalent to **jump 1.3.** It seems these calls should be grouped with the ones in 1.[3,4.5]! **Jumps:**

1.6 *Variational.h: Variational*

- 1.6.1 *instantiateCostFactory.h: instantiateCostFactory*
- 1.6.1.1 *CostFunction.h: CostMaker*
- 1.6.1.1.1 *CostFunction.h: CostFactory*

- 1.6.2 *instantiateCovarFactory.h: instantiateCovarFactory*
- 1.6.2.1 *ModelSpaceCovarianceBase.h: CovarMaker*
- 1.6.2.1.1 *ModelSpaceCovarianceBase.h: CovarianceFactory*
- 1.6.2.2 *ModelSpaceCovariance4DBase.h:* Covar4DMaker
- 1.6.2.2.1 *ModelSpaceCovariance4DBase.h:* Covariance4DFactory

1.6 *Variational.h: Variational*

- 1.6.3 *instantiateMinFactory.h: instantiateMinFactory*
- 1.6.3.1 *Minimizer.h: MinMaker*
- 1.6.3.1.1 *Minimizer.h: MinFactory*

- 1.6.4 *instantiateObsErrorFactory.h: instantiateObsErrorFactory*
- 1.6.4.1 *ObsErrorBase.h: ObsErrorMaker*
- 1.6.4.1.1 *ObsErrorBase.h: ObsErrorFactory*

- 1.6.5 *instantiateTlmFactory.h: instantiateTlmFactory*
- 1.6.5.1 *LinearModelBase.h: LinearModelMaker*
- 1.6.5.1.1 *LinearModelBase.h: LinearModelFactory*

1.7 *Run.cc: execute*

```
44    run.execute(var);
45    return 0;
46  };
```

- Here is the **main execution** of the program. **run** object has a method **execute** whose argument is of type **application. var** is an object **class Variational** and this class inherits **class Application** from namespace **eckit.** This class Application allows proper error handling of the execution...

- So that this execution leads to **1.7.1** ***Variational.h: execute***

1.7.1 *Variational.h: execute*

```
58  // ------------------------------------------------------------------------------
59    int execute(const eckit::Configuration & fullConfig) const {
60  //  Setup resolution
61      const eckit::LocalConfiguration resolConfig(fullConfig, "resolution");
62      const Geometry_ resol(resolConfig);
63
64  //  Setup Model
65      const eckit::LocalConfiguration modelConfig(fullConfig, "model");
66      const Model_ model(resol, modelConfig);
67      Log::trace() << "Variational: model has been set up" << std::endl;
68
69  /// The background is constructed inside the cost function because its valid
70  /// time within the assimilation window can be different (3D-Var vs. 4D-Var),
71  /// it can be 3D or 4D (strong vs weak constraint), etc...
72
73  //  Setup cost function
74      const eckit::LocalConfiguration cfConf(fullConfig, "cost_function");
75      boost::scoped_ptr<CostFunction<MODEL> > J(
76          CostFactory<MODEL>::create(cfConf, resol, model));
77      Log::trace() << "Variational: cost function has been set up" << std::endl;
78
79  //  Initialize first guess from background
80      ControlVariable<MODEL> xx(J->jb().getBackground());
81      Log::trace() << "Variational: first guess has been set up" << std::endl;
82
83  //  Perform Incremental Variational Assimilation
84      IncrementalAssimilation<MODEL>(xx, *J, fullConfig);
85      Log::trace() << "Variational: incremantal assimilation done" << std::endl;
86
87  //  Save analysis and final diagnostics
88      if (!fullConfig.has("mimic_masterodb_oman")) {
89        PostProcessor<State > post;
```

1.7.1.1 Setup geometry

```
58 // --------------------------------------------------------------------------
59   int execute(const eckit::Configuration & fullConfig) const {
60 //  Setup resolution
61     const eckit::LocalConfiguration resolConfig(fullConfig, "resolution");
62     const Geometry_ resol(resolConfig);
63
```

- **L61. resolConfig** is an eckit object that stores the OOPS configuration on resolution (from OOPS.json)
- **L62** defines **geometry** and **reserves memory** for it. **resol** is a instantiated **const** object of **class GeometryIFS** (see next slides)
- **Jump to 1.7.1.1.1 *Geometry.h: Geometry***

1.7.1.1 *Geometry.h: Geometry*

```
55
56  template <typename MODEL>
57  Geometry<MODEL>::Geometry(const eckit::Configuration & conf): geom_() {
58    Log::trace() << "Geometry<MODEL>::Geometry starting" << std::endl;
59    util::Timer timer(classname(), "Geometry");
60    geom_.reset(new Geometry_(conf));
61    Log::trace() << "Geometry<MODEL>::Geometry done" << std::endl;
62  }
```

- **Template** where **MODEL=IFS**, here the class **Geometry_** is an alias to the class **GeometryIFS** (see the header of Geometry.h)
- **new** is the C++ **allocate** F90 equivalent command. It reserves memory for the class Geometry_ and returns a pointer stored in **geom_**. This jumps to class GeometryIFS...

- **Jump to 1.7.1.1.1.1 *GeometryIFS.h: GeometryIFS***

1.7.1.1.1 *GeometryIFS.h: GeometryIFS*

```
46
47   explicit GeometryIFS(const eckit::Configuration & conf): geom_(new GeometryF90(conf)) {}
```

- This is the **constructor** for the object instantied in the slide before. And the constructor list call new class **GeometryF90**, whose constructor is:

```
22
23   explicit GeometryF90(const eckit::Configuration & conf) {
24     const eckit::Configuration * configc = &conf;
25     ifs_geo_setup_f90(f90geom_, &configc);
26   }
```

- **Jump to 1.7.1.1.1.1 *GeometryIFS.h: GeometryF90***
- **Jump to 1.7.1.1.1.1.1 *ifs_geo_setup_f90***

1.7.1.1.1.1.1 *ifs_geo_setup_c*:

```
37  call geometry_list%add(c_self,template)
38  self=>get_geometry(c_self)
39  ll_alternative=yconfig%has("is_alternative")
40  if(.not.ll_alternative) then
41    call geometry_setup(self, clnamelist, clorofile)
42  else
43    call geometry_setup(self, clnamelist, clorofile, ksupersede=0)
44  endif
45
```

- Eventually we call the F90 IFS code **GEOMETRY_SETUP.F90**.

1.7.1.2 Setup model

```
64  //  Setup Model
65      const eckit::LocalConfiguration modelConfig(fullConfig, "model");
66      const Model_ model(resol, modelConfig);
67      Log::trace() << "Variational: model has been set up" << std::endl;
```

- Having the resolution (resol) we instantiate the **object model.** The workflow is equivalent to **object resol:**

- **1.7.1.2 *Model.h: Model***
- **1.7.1.2.1 *ModelIFS.cc: ModelIFS***
- **1.7.1.2.1.1 *ifs_setup_f90***, which calls **MODEL_CREATE.F90**

1.7.1.3 Cost Function Factory

```
73  //  Setup cost function
74      const eckit::LocalConfiguration cfConf(fullConfig, "cost_function");
75      boost::scoped_ptr<CostFunction<MODEL> > J(
76          CostFactory<MODEL>::create(cfConf, resol, model));
77      Log::trace() << "Variational: cost function has been set up" << std::endl;
```

- We see the Factory design for CostFunction. This means there is a ***instantiateCostFactory.h*** that has several entries for cost function types. The **create method** means these optional Js are compared with string **cost_type** in json file and finally the proper J object is created. In our case the object created is from class **CostFct3Dvar**

1.7.1.3 Cost Function Factory

- **1.7.1.3 *CostFunction.h: create***
- **1.7.1.3.1 *CostFunction.h: make***
- **1.7.1.3.1.1 *CostFct3DVar.h: CostFct3DVar***
- **1.7.1.3.1.1.1 *CostFunction.h: setupTerms***

- And this **setupTerms** method goes sequentially to the **different J terms: Jo, Ji** (if present in json file?)**, Jb and Jc**. Next slide…

1.7.1.3.1.1.1 *CostFunction.h: setupTerms*

```
207 void CostFunction<MODEL>::setupTerms(const eckit::Configuration & config) {
208   Log::trace() << "CostFunction: setupTerms starting" << std::endl;
209
210 // Jo
211   std::vector<eckit::LocalConfiguration> jos;
212   config.get("Jo", jos);
213   for (size_t jj = 0; jj < jos.size(); ++jj) {
214     CostJo<MODEL> * jo = this->newJo(jos[jj]);
215     jterms_.push_back(jo);
216   }
```

- For loop in **L213** is expecting **any Ji diferent than Jb**. It seems this should be set through json file... In our case **only 1 iteration, only Jo.**
- **L214**, jumps to:
- **1.7.1.3.1.1.1.1 *CostFct3DVar.h: newJo***
- **1.7.1.3.1.1.1.1.1.1 *CostJo.h: CostJo,*** next slide…

1.7.1.3.1.1.1.1.1 *CostJo.h: CostJo*

```
132 CostJo<MODEL>::CostJo(const eckit::Configuration & joConf,
133                       const util::DateTime & winbgn, const util::DateTime & winend,
134                       const util::Duration & ts, const bool subwindows)
135   : obspace_(eckit::LocalConfiguration(joConf, "Observation"), winbgn, winend),
136     hop_(obspace_, eckit::LocalConfiguration(joConf, "Observation")),
137     yobs_(obspace_),
138     R_(obspace_, eckit::LocalConfiguration(joConf, "Covariance")),
139     gradFG_(), pobs_(), tslot_(ts),
140     hoptlad_(), subwindows_(subwindows), ltraj_(false)
141 {
142   Log::debug() << "CostJo:setup tslot_ = " << tslot_ << std::endl;
143   yobs_.read(eckit::LocalConfiguration(joConf, "Observation"));
144   Log::trace() << "CostJo:CostJo done" << std::endl;
145 }
```

- The main work is done in the ***constructor list.*** So, all the objects: **obspace_, hop_, yobs_, R_, gradFG_, pobs_, tslot_, hoptlad_, suwindows_, ltraj_** of Jo are stored in the private part of the class CostJo **(encapsulation!).**
- Let's start with **obspace_** next slide

1.7.1.3.1.1.1.1.1.1 *ObservationSpace.h: ObservationSpace*

- This jumps to **1.7.1.3.1.1.1.1.1.1.1** ***ObsSpaceODB.cc: ObsSpaceODB***
- **1.7.1.3.1.1.1.1.1.1.1.1** ***odb_setup_f90,*** which calls procedure **SETUP** from **OBS_SPACE_MOD.F90**

```
16  ObsSpaceODB::ObsSpaceODB(const eckit::Configuration & config,
17                           const util::DateTime & bgn, const util::DateTime & end)
18    : winbgn_(bgn), winend_(end), winlen_(end - bgn)
19  {
20    const GeometryIFS geom(eckit::LocalConfiguration(config, "local_geom_obs_space"));
21    if (nbodb_ == 0) {
22      const eckit::Configuration * configc = &config;
23      odb_setup_f90(theodb_, geom.toFortran(), winlen_.toSeconds(), &configc);
24    }
25    f90odb_ = theodb_;
26    nbodb_ += 1;
27    Log::trace() << "ObsSpaceODB constructed" << std::endl;
```

1.7.1.3.1.1.1.1.1 *CostJo.h: CostJo*

```
132 CostJo<MODEL>::CostJo(const eckit::Configuration & joConf,
133                       const util::DateTime & winbgn, const util::DateTime & winend,
134                       const util::Duration & ts, const bool subwindows)
135   : obspace_(eckit::LocalConfiguration(joConf, "Observation"), winbgn, winend),
136     hop_(obspace_, eckit::LocalConfiguration(joConf, "Observation")),
137     yobs_(obspace_),
138     R_(obspace_, eckit::LocalConfiguration(joConf, "Covariance")),
139     gradFG_(), pobs_(), tslot_(ts),
140     hoptlad_(), subwindows_(subwindows), ltraj_(false)
141 {
142   Log::debug() << "CostJo:setup tslot_ = " << tslot_ << std::endl;
143   yobs_.read(eckit::LocalConfiguration(joConf, "Observation"));
144   Log::trace() << "CostJo:CostJo done" << std::endl;
145 }
```

- Time for **hop_**. Jumps to
- **1.7.1.3.1.1.1.1.1.2 *ObsOperator.h: ObsOperator***
- **1.7.1.3.1.1.1.1.1.2.1 *AllObs.cc: AllObs***

1.7.1.3.1.1.1.1.1.2.1 *AllObs.cc: AllObs*

```
21  AllObs::AllObs(ObsSpaceODB & odb) : f90oper_(0), odb_(odb), inputs_() {
22    obs_oper_setup_f90(f90oper_,odb_.toFortran());
23    // Temporary hack for defining variables. YT
24  //  const long ilin = 0;  // 1 is for linear
25  //  const long ictl = 0;  // 1 is for control variable only
26  //  eckit::LocalConfiguration vars;
27  //  vars.set("linear", ilin);
28  //  vars.set("control", ictl);
29    inputs_.reset(new VariablesIFS(0, 0));
30    // Temporary hack for defining variables. YT
31    Log::trace() << "AllObs constructed" << std::endl;
```

- **L22. 1.7.1.3.1.1.1.1.1.2.1.1 obs_oper_setup_f90,** calls the F90 routine **OBS_OPER_SETUP.F90**
- **L29** VariablesIFS(0,0). Jumps**:**
- **1.7.1.3.1.1.1.1.1.2.1.2 *VariablesIFS.h: VariablesIFS***
- **1.7.1.3.1.1.1.1.1.2.1.2.1 *ifs_variables_setup_f90,*** that calls **VARIABLES_CREATE.F90**

1.7.1.3.1.1.1.1.1 *CostJo.h: CostJo*

```
132  CostJo<MODEL>::CostJo(const eckit::Configuration & joConf,
133                        const util::DateTime & winbgn, const util::DateTime & winend,
134                        const util::Duration & ts, const bool subwindows)
135    : obspace_(eckit::LocalConfiguration(joConf, "Observation"), winbgn, winend),
136      hop_(obspace_, eckit::LocalConfiguration(joConf, "Observation")),
137      yobs_(obspace_),
138      R_(obspace_, eckit::LocalConfiguration(joConf, "Covariance")),
139      gradFG_(), pobs_(), tslot_(ts),
140      hoptlad_(), subwindows_(subwindows), ltraj_(false)
141  {
142    Log::debug() << "CostJo:setup tslot_ = " << tslot_ << std::endl;
143    yobs_.read(eckit::LocalConfiguration(joConf, "Observation"));
144    Log::trace() << "CostJo:CostJo done" << std::endl;
145  }
```

- L137. Time for **yobs_**. Jump to **1.7.1.3.1.1.1.1.1.3 *Observations.h: Observations***
- This object (observations) is filled with data read when constructing **obspace_**, so no call to any F90 routine. Seems geometry of obspace

1.7.1.3.1.1.1.1.1 *CostJo.h: CostJo*

```
132 CostJo<MODEL>::CostJo(const eckit::Configuration & joConf,
133                       const util::DateTime & winbgn, const util::DateTime & winend,
134                       const util::Duration & ts, const bool subwindows)
135   : obspace_(eckit::LocalConfiguration(joConf, "Observation"), winbgn, winend),
136     hop_(obspace_, eckit::LocalConfiguration(joConf, "Observation")),
137     yobs_(obspace_),
138     R_(obspace_, eckit::LocalConfiguration(joConf, "Covariance")),
139     gradFG_(), pobs_(), tslot_(ts),
140     hoptlad_(), subwindows_(subwindows), ltraj_(false)
141 {
142   Log::debug() << "CostJo:setup tslot_ = " << tslot_ << std::endl;
143   yobs_.read(eckit::LocalConfiguration(joConf, "Observation"));
144   Log::trace() << "CostJo:CostJo done" << std::endl;
145 }
```

- **L138.** Time for **R_**. Jump to **1.7.1.3.1.1.1.1.1.4 *ObsErrorCovariance.h: ObsErrorCovariance***

- ObsError Factory (see ***InstantiateObsErrorFactory.*h**)

1.7.1.3.1.1.1.1.1.4.1 *Obs Error Factory*

- **1.7.1.3.1.1.1.1.1.4.1 *ObsErrorBase.h: create***
- **1.7.1.3.1.1.1.1.1.4.1.1 *ObsErrorBase.h: make***
- **1.7.1.3.1.1.1.1.1.4.1.1.1: *AllObsCovariance.cc: AllObsCovariance***
- **1.7.1.3.1.1.1.1.1.4.1.1.1.1 *ifs_r_setup_f90,*** that calls **R_CREATE.F90**

```
18  AllObsCovariance::AllObsCovariance(const ObsSpaceODB & odb,
19                                     const eckit::Configuration & conf) {
20    const eckit::Configuration * config = &conf;
21    ifs_r_setup_f90(self_, odb.toFortran(), &config);
22    Log::trace() << "AllObsCovariance created" << std::endl;
23  }
```

1.7.1.3.1.1.1.1.1 *CostJo.h: CostJo*

```
132  CostJo<MODEL>::CostJo(const eckit::Configuration & joConf,
133                        const util::DateTime & winbgn, const util::DateTime & winend,
134                        const util::Duration & ts, const bool subwindows)
135    : obspace_(eckit::LocalConfiguration(joConf, "Observation"), winbgn, winend),
136      hop_(obspace_, eckit::LocalConfiguration(joConf, "Observation")),
137      yobs_(obspace_),
138      R_(obspace_, eckit::LocalConfiguration(joConf, "Covariance")),
139      gradFG_(), pobs_(), tslot_(ts),
140      hoptlad_(), subwindows_(subwindows), ltraj_(false)
141  {
142    Log::debug() << "CostJo:setup tslot_ = " << tslot_ << std::endl;
143    yobs_.read(eckit::LocalConfiguration(joConf, "Observation"));
144    Log::trace() << "CostJo:CostJo done" << std::endl;
145  }
```

- **L139-L140. gradFG_**, **pobs_** and **hoptlad_** are pointers initialized (boost scoped_ptr/shared_ptr). **tslot_** is initialized with input object **ts** and **subwindows_** and **ltraj_** with boolean inputs. So no call to any F90 routine!
- **L143 yobs_.read** reads **observations!** Next slide...

1.7.1.3.1.1.1.1.1.1 *CostJo.h: CostJo*

- Jump to **1.7.1.3.1.1.1.1.1.1.5 *Observations.h: read***
- Jump to **1.7.1.3.1.1.1.1.1.1.5.1 *ObsVector.h: read***
- Jump to **1.7.1.3.1.1.1.1.1.1.5.1.1 *ObsVector.cc: read***
- **1.7.1.3.1.1.1.1.1.1.5.1.1.1 *ifs_obsvec_read_f90,*** that calls **READ_OBSVEC.F90**

- With this **Jo constructed!**

1.7.1.3.1.1.1 *CostFunction.h: setupTerms*

```
219  // Jb
220    const eckit::LocalConfiguration jbConf(config, "Jb");
221    xb_.reset(new CtrlVar_(eckit::LocalConfiguration(jbConf, "Background"), resol_, model_));
222    jb_.reset(new JbTotal_(*xb_, this->newJb(jbConf, resol_, *xb_), jbConf, resol_));
223    Log::trace() << "CostFunction: setupTerms Jb added" << std::endl;
224
```

- L221. Time for **Jb**. **xb_** is a scoped_ptr of class **ControlVariable**. Method **reset** is from class **scoped_ptr**. If one want to access methods of **xb_** one should **dereference it: *xb_**. In particular **reset** frees memory and gives the content of () to the new pointer **xb_**.
- Jump to **1.7.1.3.1.1.1.2 *ControlVariable.h: ControlVariable, n*ext slide

1.7.1.3.1.1.1.2 *ControlVariable.h: ControlVariable*

```
88  ControlVariable<MODEL>::ControlVariable(const eckit::Configuration & conf,
89                                          const Geometry_ & resol, const Model_ & model)
90    : state4d_(conf, resol, model),
91      modbias_(resol, conf.getSubConfiguration("ModelBias")),
92      obsbias_(conf.getSubConfiguration("ObsBias"))
93  {
94    Log::trace() << "ControlVariable contructed" << std::endl;
```

- Here the work again is done in the ***constructor list.***
- **state4d_** reads the FA file, the background, at the beginning of AW.
- **1.7.1.3.1.1.1.2.1 *State4D.h: State4D***
- **1.7.1.3.1.1.1.2.1.1 *State.h: State***
- **1.7.1.3.1.1.1.2.1.1.1 *StateIFS.cc: StateIFS,*** next slide

1.7.1.3.1.1.1.2.1.1.1 *StateIFS.cc: StateIFS*

```
34  // ------------------------------------------------------------------------
35    StateIFS::StateIFS(const GeometryIFS & resol, const ModelIFS & model, const eckit::
36    : fields_(), vars_(), reftime_(), time_()
37  {
38  // Should get variables from file. YT
39    vars_.reset(new VariablesIFS(file));
40  // Should get variables from file. YT
41
42    fields_.reset(new FieldsIFS(resol, *vars_, model));
43    util::Duration step(0);
44    fields_->read(file, reftime_, step);
45    time_ = reftime_ + step;
46
47    Log::trace() << "StateIFS::StateIFS created and read in." << std::endl;
```

- L39. In json file keyword **variables** controls which variables will be present in the Control State.
- Jump to **1.7.1.3.1.1.1.2.1.1.1.1 *VariablesIFS.h: VariablesIFS***
- **1.7.1.3.1.1.1.2.1.1.1.1.1 *ifs_variables_create_f90,*** which calls **VARIABLES_CREATE.F90**

1.7.1.3.1.1.1.2.1.1.1 *StateIFS.cc: StateIFS*

```
34 // ------------------------------------------------------------------------
35   StateIFS::StateIFS(const GeometryIFS & resol, const ModelIFS & model, const eckit::
36   : fields_(), vars_(), reftime_(), time_()
37 {
38 // Should get variables from file. YT
39   vars_.reset(new VariablesIFS(file));
40 // Should get variables from file. YT
41
42   fields_.reset(new FieldsIFS(resol, *vars_, model));
43   util::Duration step(0);
44   fields_->read(file, reftime_, step);
45   time_ = reftime_ + step;
46
47   Log::trace() << "StateIFS::StateIFS created and read in." << std::endl;
```

- **L42**. **Fields are created** taking resolution, variables and model.
- Jump to **1.7.1.3.1.1.1.2.1.1.1.2 *FieldsIFS.cc: FieldsIFS***
- **1.7.1.3.1.1.1.2.1.1.1.2.1 *ifs_field_create_f90*** (This routine calls **FIELDS_CREATE.F90**)

1.7.1.3.1.1.1.2.1.1.1 *StateIFS.cc: StateIFS*

```
34  // ------------------------------------------------------------------------
35    StateIFS::StateIFS(const GeometryIFS & resol, const ModelIFS & model, const eckit::
36    : fields_(), vars_(), reftime_(), time_()
37  {
38  // Should get variables from file. YT
39    vars_.reset(new VariablesIFS(file));
40  // Should get variables from file. YT
41
42    fields_.reset(new FieldsIFS(resol, *vars_, model));
43    util::Duration step(0);
44    fields_->read(file, reftime_, step);
45    time_ = reftime_ + step;
46
47    Log::trace() << "StateIFS::StateIFS created and read in." << std::endl;
```

- **L44. Read fields from FA background file...**
- Jump to **1.7.1.3.1.1.1.2.1.1.1.3 *FieldsIFS.cc: read***
- **1.7.1.3.1.1.1.2.1.1.1.3.1 *ifs_field_read_file_f90*** and this calls **READ_FIELDS.F90**

1.7.1.3.1.1.1.2 *ControlVariable.h: ControlVariable*

```
88  ControlVariable<MODEL>::ControlVariable(const eckit::Configuration & conf,
89                                          const Geometry_ & resol, const Model_ & model)
90    : state4d_(conf, resol, model),
91      modbias_(resol, conf.getSubConfiguration("ModelBias")),
92      obsbias_(conf.getSubConfiguration("ObsBias"))
93  {
94    Log::trace() << "ControlVariable contructed" << std::endl;
```

- **modbias_** in the ControVariable (model error!, weak constraint...) After checking, in Cy46, we can see that **modbias** is **not implemented**.
- Jump to **1.7.1.3.1.1.1.2.2 *ModelBias.h: ModelBias* (not implemented)**
- It has been checked that **in Cy49 this is very well developed!**

1.7.1.3.1.1.1.2 *ControlVariable.h: ControlVariable*

```
88  ControlVariable<MODEL>::ControlVariable(const eckit::Configuration & conf,
89                                          const Geometry_ & resol, const Model_ & model)
90    : state4d_(conf, resol, model),
91      modbias_(resol, conf.getSubConfiguration("ModelBias")),
92      obsbias_(conf.getSubConfiguration("ObsBias"))
93  {
94    Log::trace() << "ControlVariable contructed" << std::endl;
```

- **L92. obsbias_** in the ControlVariable... This is what deals with **VARBC**
- Jump to **1.7.1.3.1.1.1.2.3 *ObsAuxControl.h: ObsAuxControl***
- Jump to **1.7.1.3.1.1.1.2.3.1 *ObsBias.cc: ObsBias***
- **1.7.1.3.1.1.1.2.3.1.1 *ifs_obs_bias_setup_f90,*** that calls **SETUP_TRAJ** procedure from **VARBC_CLASS.F90**

1.7.1.3.1.1.1.2 *ControlVariable.h: ControlVariable*

```
88  ControlVariable<MODEL>::ControlVariable(const eckit::Configuration & conf,
89                                          const Geometry_ & resol, const Model_ & model)
90    : state4d_(conf, resol, model),
91      modbias_(resol, conf.getSubConfiguration("ModelBias")),
92      obsbias_(conf.getSubConfiguration("ObsBias"))
93  {
94    Log::trace() << "ControlVariable contructed" << std::endl;
```

- **1.7.1.3.1.1.1.2.3.1.2 *obsbias_tcv_setup_f90,*** that calls **CREATE_TOVSCV** procedure from **TOVSCV_MOD.F90**
- **1.7.1.3.1.1.1.2.3.1.3 *obsbias_tcv_read_f90 ,*** that calls **READ_TOVSCV** procedure from **TOVSCV_MOD.F90**

1.7.1.3.1.1.1 *CostFunction.h: setupTerms*

```
219  // Jb
220    const eckit::LocalConfiguration jbConf(config, "Jb");
221    xb_.reset(new CtrlVar_(eckit::LocalConfiguration(jbConf, "Background"), resol_, model_));
222    jb_.reset(new JbTotal_(*xb_, this->newJb(jbConf, resol_, *xb_), jbConf, resol_));
223    Log::trace() << "CostFunction: setupTerms Jb added" << std::endl;
224
```

- With this ControlVariable is constructed. Next **Total Jb…**
- Jump to **1.7.1.3.1.1.1.3 *CostFct3DVar.h: newJb***
- Jump to **1.7.1.3.1.1.1.3.1 *CostJb3D.h: CostJb3D,*** next slide…

1.7.1.3.1.1.1.3.1 *CostJb3D.h: CostJb3D*

```
126 CostJb3D<MODEL>::CostJb3D(const eckit::Configuration & config, const Geometry_ & resolouter,
127                           const Variables_ & ctlvars, const util::Duration & len,
128                           const State_ & xb)
129   : B_(CovarianceFactory<MODEL>::create(eckit::LocalConfiguration(config, "Covariance"),
130                                         resolouter, ctlvars, xb)),
131     winLength_(len), controlvars_(ctlvars), resol_(), time_()
132 {
133   Log::trace() << "CostJb3D constructed." << std::endl;
134 }
```

- Work is **constructing B_** which again is linked to the **Factory Covariance**, present in ***InstantiateCovarFactory.h*****,** trace:

- **1.7.1.3.1.1.1.3.1.1 *ModelSpaceCovarianceBase.h: create***
- **1.7.1.3.1.1.1.3.1.1.1 *ModelSpaceCovarianceBase.h: make***
- **1.7.1.3.1.1.1.3.1.1.1.1 *ErrorCovariance.h: ErrorCovariance***

1.7.1.3.1.1.1.3.1 *CostJb3D.h: CostJb3D*

- **1.7.1.3.1.1.1.3.1.1.1.1.1 *ErrorCovariance3D.cc: ErrorCovariance3D***

```
20  ErrorCovariance3D::ErrorCovariance3D(const GeometryIFS & resol, const Variables
21                                       const eckit::Configuration & conf, const S
22    : ctrlvars_(cvars), config_(conf), xb_(xb)
23  {
24    time_ = util::DateTime(conf.getString("date"));
25    const eckit::Configuration * configc = &conf;
26    ifs_error_covariance_3d_setup_f90(ftnSelf_, &configc, resol.toFortran());
27    Log::trace() << "ErrorCovariance3D created" << std::endl;
28  }
```

- **L22. In ctrlvars_(cvars), jumps:**
- **1.7.1.3.1.1.1.3.1.1.1.1.1.1 *VariablesIFS.h: VariablesIFS***
- **1.7.1.3.1.1.1.3.1.1.1.1.1.1.1 *ifs_variables_clone_f90,*** which calls **VARIABLES_CLONE.F90**

1.7.1.3.1.1.1.3.1 *CostJb3D.h: CostJb3D*

- **1.7.1.3.1.1.1.3.1.1.1.1.1 *ErrorCovariance3D.cc: ErrorCovariance3D***

```
20  ErrorCovariance3D::ErrorCovariance3D(const GeometryIFS & resol, const Variables
21                                       const eckit::Configuration & conf, const S
22    : ctrlvars_(cvars), config_(conf), xb_(xb)
23  {
24    time_ = util::DateTime(conf.getString("date"));
25    const eckit::Configuration * configc = &conf;
26    ifs_error_covariance_3d_setup_f90(ftnSelf_, &configc, resol.toFortran());
27    Log::trace() << "ErrorCovariance3D created" << std::endl;
28  }
```

- **L26. 1.7.1.3.1.1.1.3.1.1.1.1.1.2 *ifs_error_covariance_3d_setup_f90,*** which calls **CREATE** procedure from **ERROR_COVARIANCE_3D_MOD.F9**

1.7.1.3.1.1.1 *CostFunction.h: setupTerms*

```
225  // Other constraints
226    std::vector<eckit::LocalConfiguration> jcs;
227    config.get("Jc", jcs);
228    for (size_t jj = 0; jj < jcs.size(); ++jj) {
229      CostTermBase<MODEL> * jc = this->newJc(jcs[jj], resol_);
230      jterms_.push_back(jc);
231    }
232    Log::trace() << "CostFunction: setupTerms Jc added" << std::endl;
233    Log::trace() << "CostFunction: setupTerms done" << std::endl;
234  }
```

- Time for **Jc** (gravity wave control). Well, Jc is not present in json file and besides for 3DVAR Jc it is not implemented in CY46. See ***CostFct3DVar.h: newJc*** method. **It is for 4DVAR!!**

- With this **J is constructed** and we go back to **Variational.h**

1.7.1.4 *CostJbTotal.h: getBackground*

```
79  //  Initialize first guess from background
80      ControlVariable<MODEL> xx(J->jb().getBackground());
81      Log::trace() << "Variational: first guess has been set up" << std::endl;
82
```

- L80. This does nothing special but get the **background state** already read in Jb computation setup. Here **xx** is a pointer to the **background.** See jump **1.7.1.3.1.1.1.2.** Jump to **1.7.1.4** ***CostJbTotal.h: getBackground***

- Here **xx** is an object of ControlVariable class. **Control Variable (X)** is in a different space than **Control Increment (dx),** but dimensions are the same:

$$\delta \mathbf{x} = \mathbf{U}\chi \qquad \mathbf{B} = \mathbf{U}\mathbf{U}^T.$$

1.7.1.5 *IncrementalAssimilation.h: IncrementalAssimilation*

```
83  //  Perform Incremental Variational Assimilation
84      IncrementalAssimilation<MODEL>(xx, *J, fullConfig);
85      Log::trace() << "Variational: incremantal assimilation done" << std::endl;
86
```

- **L84. IncrementalAssimilation** is a **void method** (see IncrementalAssimilation.h) that is the framework core for the minimization algorithm. Next slide

1.7.1.5 *IncrementalAssimilation.h: IncrementalAssimilation*

```
43
44  // Setup outer loop
45    std::vector < eckit::LocalConfiguration > iterconfs;
46    config.get("variational.iteration", iterconfs);
47    unsigned int nouter = iterconfs.size();
48    Log::info() << "Running incremental assimilation with " << nouter
49        << " outer iterations." << std::endl;
50
51  // Setup minimizer
52    eckit::LocalConfiguration minConf(config, "minimizer");
53    const long nnout = nouter;
54    minConf.set("nouter", nnout);
55    boost::scoped_ptr < Minimizer_ > minim(MinFactory<MODEL>::create(minConf, J));
56
57    for (unsigned jouter = 0; jouter < nouter; ++jouter) {
58  //  Get configuration for current outer iteration
59      Log::info() << "IncrementalAssimilation: Configuration for outer iteration "
60          << jouter << ":\n" << iterconfs[jouter];
61
62  //  Setup for the trajectory run
63      PostProcessor<State_> post;
64      if (iterconfs[jouter].has("prints")) {
65        const eckit::LocalConfiguration prtConfig(iterconfs[jouter], "prints");
66        post.enrollProcessor(new StateInfo<State_>("traj", prtConfig));
67      }
68
69  //  Setup quadratic problem
70      J.linearize(xx, iterconfs[jouter], post);
71
72  //  Minimization
73      boost::scoped_ptr<CtrlInc_> dx(minim->minimize(iterconfs[jouter]));
74
75  //  Compute analysis in physical space
76      J.addIncrement(xx, *dx);
```

1.7.1.5 *IncrementalAssimilation.h: IncrementalAssimilation*

```
44  // Setup outer loop
45    std::vector < eckit::LocalConfiguration > iterconfs;
46    config.get("variational.iteration", iterconfs);
47    unsigned int nouter = iterconfs.size();
48    Log::info() << "Running incremental assimilation with " << nouter
49        << " outer iterations." << std::endl;
50
51  // Setup minimizer
52    eckit::LocalConfiguration minConf(config, "minimizer");
53    const long nnout = nouter;
54    minConf.set("nouter", nnout);
55    boost::scoped_ptr < Minimizer_ > minim(MinFactory<MODEL>::create(minConf, J));
56
```

- Firstly json block “**variational.iteration”** is read, with info on number of inner loops iterations, type of tangent linear model and variational tests... And more
- Secondly, json block “**minimizer”** is read. There is decided which linear solver will be used to solve the **∇J(X)=0** equation. In our case **SQRTPLanczos. L55** Construct **object minim. MinFactory.** Next slide.

1.7.1.5 Traceback for Minimizer Factory

- **1.7.1.5.1 *Minimizer.h: create***
- **1.7.1.5.1.1 *Minimizer.h: make***
- **1.7.1.5.1.1.1 *SQRTPLanczosMinimizer.h: SQRTPLanczosMinimizer***
- **1.7.1.5.1.1.1.1 *SQRTMinimizer.h*: *SQRTMinimizer***
- **1.7.1.5.1.1.1.1.1 *Minimizer.h: Minimizer***

- With this minimizer, **object minim** in slide before, constructed
- See ***instantiateMinFactory.h*** for available minimizers

1.7.1.5 *IncrementalAssimilation.h: IncrementalAssimilation:*

```
57    for (unsigned jouter = 0; jouter < nouter; ++jouter) {
58  //  Get configuration for current outer iteration
59      Log::info() << "IncrementalAssimilation: Configuration for outer iteration "
60          << jouter << ":\n" << iterconfs[jouter];
61
62  //  Setup for the trajectory run
63      PostProcessor<State_> post;
64      if (iterconfs[jouter].has("prints")) {
65        const eckit::LocalConfiguration prtConfig(iterconfs[jouter], "prints");
66        post.enrollProcessor(new StateInfo<State_>("traj", prtConfig));
67      }
```

- **Outer loop.** Each iteration has its own setup (resolution and others, in json file). Clear this in 4DVAR OOPS LELAM... nouter=1 in 3DVAR.
- **PostProcessor** (**PP**) **design.** Clever design to apply different PostProcessor tasks to the **forecast step**: **compute the trajectory, write a State file, compute H(x)...**
- Here we prepare PP for trajectory, declaring the object **post.** L66 writes debug info (probably), we don’t enter the if in our case...

1.7.1.5 *IncrementalAssimilation.h: IncrementalAssimilation:*

```
69    //  Setup quadratic problem
70        J.linearize(xx, iterconfs[jouter], post);
71
```

- 3 main jobs done here: **getting the LR trajectory** along the AW, the **pseudobservations H(x) where x is in HR** to compute departures and **compute scalar value of Cost Function J**. These 3 jobs are updated in each outer loop...

- **Jump to 1.7.1.5.2 *CostFunction.h: linearize***

1.7.1.5.2 *CostFunction.h: linearize*

```
273  // Read inner loop configuration
274    const eckit::LocalConfiguration resConf(innerConf, "resolution");
275    const eckit::LocalConfiguration tlmConf(innerConf, "linearmodel");
276    eckit::LocalConfiguration diagnostic;
277    if (innerConf.has("diagnostics")) {
278      diagnostic = eckit::LocalConfiguration(innerConf, "diagnostics");
279    }
280    const Geometry_ lowres(resConf);
```

- First read diferents parts of configuration and declare a **lowres** object with **the geometry of the current inner loop.** L280 trace:

- **1.7.1.5.2.1 *Geometry.h: Geometry***
- **1.7.1.5.2.1.1 *GeometryIFS.h: GeometryIFS***
- **1.7.1.5.2.1.1.1 *GeometryIFS.h: GeometryF90***
- **1.7.1.5.2.1.1.1.1 *ifs_geo_setup_c,*** calling **GEOMETRY_SETUP.F90**

1.7.1.5.2 *CostFunction.h: linearize*

```
282  // Setup terms of cost function
283    PostProcessor<State_> pp(post);
284    JqTerm_ * jq = jb_->initializeTraj(fguess, lowres);
285    pp.enrollProcessor(jq);
286    for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
287      pp.enrollProcessor(jterms_[jj].initializeTraj(fguess, lowres, innerConf));
288    }
289
```

- Prepare the diferent **PostProcessors** that will be applied to the forecat step. L283 **copies input post** to new declared object **pp.** Not sure what this implies...
- L284 **initialize Jb** gets the **trajectory** and the corrected-with-first guess **B matrix** (see 5 slides ahead) at the **beginning of AW** both in **low resolution Jump to 1.7.1.5.2.2....**
- It is **confusing** to use the class **JqTerm** since **we are not dealing with Jq....**

1.7.1.5.2.2 *CostJbTotal.h: initializeTraj*

```
180   fg_ = &fg;
181   Log::debug() << "CostJbTotal:initializeTraj first guess is:" << *fg_ << std::endl;
182   resol_.reset(new Geometry_(resol));
183 // Linearize terms
184   jb_->linearize(fg.state(), *resol_);
185   jbModBias_.linearize(fg.modVar(), *resol_);
186   jbObsBias_.linearize(fg.obsVar());
```

- L184. **jb_** is of class **CostJbState.** This class has a method **linearize** declared as **virtual.** This means that this method can be modified in a **derived class (inheritance).** In **CostJbState.h:**

```
69  /// Linearize before the linear computations.
70    virtual void linearize(const State4D_ &, const Geometry_ &) =0;
71
```

- In our case we jump to **1.7.1.5.2.2.1 *CostJb3D.h: linearize***

1.7.1.5.2.2.1 *CostJb3D.h: linearize*

```
139 void CostJb3D<MODEL>::linearize(const State4D_ & fg, const Geometry_ & resolinner) {
140   ASSERT(fg.checkStatesNumber(1));
141   resol_.reset(new Geometry_(resolinner));
142   time_.reset(new util::DateTime(fg[0].validTime()));
143   B_->linearize(fg[0], *resol_);
144 }
```

- L143 **jumps:**
- **to 1.7.1.5.2.2.1.1 *ErrorCovariance.h: linearize***
- **to 1.7.1.5.2.2.1.1.1 *ErrorCovariance3D.cc: linearize***

1.7.1.5.2.2.1.1.1 *ErrorCovariance3D.cc: linearize*

```
38 void ErrorCovariance3D::linearize(const StateIFS & fg, const GeometryIFS & resol) {
39   geom_.reset(new GeometryIFS(resol));
40   const eckit::Configuration * configc = &config_;
41   //StateIFS fgtmp(fg);
42   fglr_.reset(new StateIFS(resol,fg));
43   //StateIFS xbtmp(xb_);
44   xblr_.reset(new StateIFS(resol,xb_));
45   ifs_error_covariance_3d_linearize_f90(ftnSelf_, xblr_->toFortran(),fglr_->toFortran(), geom_->toFortran(),
46 }
```

- Here in L42 and L44 we prepare **first guess and background** in **inner loop resolution...**

- **L42 jump to 1.7.1.5.2.2.1.1.1.1 *StateIFS.cc: StateIFS***
- **Jump to 1.7.1.5.2.2.1.1.1.1.1 *ModelIFS.cc: FieldsIFS***
- **Jump to 1.7.1.5.2.2.1.1.1.1.1.1.1 *ifs_field_create_c,*** that calls **FIELDS_CREATE.F90**

1.7.1.5.2.2.1.1.1 *ErrorCovariance3D.cc: linearize*

```
38 void ErrorCovariance3D::linearize(const StateIFS & fg, const GeometryIFS & resol) {
39   geom_.reset(new GeometryIFS(resol));
40   const eckit::Configuration * configc = &config_;
41   //StateIFS fgtmp(fg);
42   fglr_.reset(new StateIFS(resol,fg));
43   //StateIFS xbtmp(xb_);
44   xblr_.reset(new StateIFS(resol,xb_));
45   ifs_error_covariance_3d_linearize_f90(ftnSelf_, xblr_->toFortran(),fglr_->toFortran(), geom_->toFortran(),
46 }
```

- Here in L42 and L44 we prepare **first guess and background** in **inner loop resolution...**

- **L42 jump to 1.7.1.5.2.2.1.1.1.1 *StateIFS.cc: StateIFS***
- **Jump to 1.7.1.5.2.2.1.1.1.1.2 *FieldsIFS.cc: changeResolution***
- **Jump to 1.7.1.5.2.2.1.1.1.1.2.1 *ifs_field_change_resol_f90,*** that calls **FIELDS_FP_CHGE_RESOL.F90**

1.7.1.5.2.2.1.1.1 *ErrorCovariance3D.cc: linearize*

```
38  void ErrorCovariance3D::linearize(const StateIFS & fg, const GeometryIFS & resol) {
39    geom_.reset(new GeometryIFS(resol));
40    const eckit::Configuration * configc = &config_;
41    //StateIFS fgtmp(fg);
42    fglr_.reset(new StateIFS(resol,fg));
43    //StateIFS xbtmp(xb_);
44    xblr_.reset(new StateIFS(resol,xb_));
45    ifs_error_covariance_3d_linearize_f90(ftnSelf_, xblr_->toFortran(),fglr_->toFortran(), geom_->toFortran(),
46  }
```

- **L44. 1.7.1.5.2.2.1.1.1.2** repeats exactly the **same traceback** than **L42** but for **background!**
- **L45. 1.7.1.5.2.2.1.1.1.3 *ifs_error_covariance_3d_linearize_f90,*** which calls procedure **LINEARIZE** from **ERROR_COVARIANCE_3D_MOD.F90.** Not sure what happens here theoretically… It seems there is some modification of B matrix with differences of background and first guess… ??

1.7.1.5.2.2 *CostJbTotal.h: initializeTraj*

```
183  // Linearize terms
184    jb_->linearize(fg.state(), *resol_);
185    jbModBias_.linearize(fg.modVar(), *resol_);
186    jbObsBias_.linearize(fg.obsVar());
187
188    JqTerm_ * jqnl = this->initialize(fg);
189    if (jqnl) jqnl->linearize();
190    return jqnl;
191  }
```

- L185 and L186 don’t do much. They are related to **model error and obs bias** parts of **control vector** and they are not implemented in CY46.
- L188 **jqnl** returns a null pointer **jqnl** returns also a **null pointer: Jq is not implemented! (Cy46).** We then go back to **CostFunction.h.**

1.7.1.5.2 *CostFunction.h: linearize*

```
282  // Setup terms of cost function
283    PostProcessor<State_> pp(post);
284    JqTerm_ * jq = jb_->initializeTraj(fguess, lowres);
285    pp.enrollProcessor(jq);
286    for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
287      pp.enrollProcessor(jterms_[jj].initializeTraj(fguess, lowres, innerConf));
288    }
289
```

- L285, each new processor is represented by a **pointer** (here **jq)** that is **pushed back (attach in C++)** by method **enrollProcessor** (see PostProcessor.h) to construct a **list of processors** in object **pp**
- L286 loop is over size of **jterms_,** which is a private member of the own class CostFunction and it has been previously computed. It has term **Jo** only in our case, so 1 iteration

1.7.1.5.2 *CostFunction.h: linearize*

```
282  // Setup terms of cost function
283    PostProcessor<State_> pp(post);
284    JqTerm_ * jq = jb_->initializeTraj(fguess, lowres);
285    pp.enrollProcessor(jq);
286    for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
287      pp.enrollProcessor(jterms_[jj].initializeTraj(fguess, lowres, innerConf));
288    }
289
```

- L287, add processor related to **Jo.** This will compute, **H(x), H^T(x),** in each observation window time...

- **Jump to 1.7.1.5.2.3 *CostJo.h: initializeTraj***

1.7.1.5.2.3 *CostJo.h: initializeTraj*

```
187  CostJo<MODEL>::initializeTraj(const CtrlVar_ & xx, const Geometry_ & lrgeom,
188                                const eckit::Configuration & config) {
189    ltraj_ = true;
190    hoptlad_.reset(new LinearObsOperator_(hop_));
191    if (config.has("lrhtraj")) {
```

- L190, construct the **object H^T**, so adjoint of **H. H^T** is constructed taking the values of **H** as initial guess, reordering them (transposing) in the appropriate **AllObsTLAD class.** Jumps:
- **1.7.1.5.2.3.1 *LinearObsOperator.h: LinearObsOperator***
  1.7.1.5.2.3.1.1 *AllObsTLAD.h: create*
- **1.7.1.5.2.3.1.1.1 *AllObsTLAD.cc: AllObsTLAD***
  1.7.1.5.2.3.1.1.1.1 *VariablesIFS.h: VariablesIFS*
  1.7.1.5.2.3.1.1.1.1.1 *ifs_variables_setup_f90,* calling **VARIABLES_CREATE.F90**

1.7.1.5.2.3 *CostJo.h: initializeTraj*

```
203     Log::info() << "CostJo<MODEL>::initializeTraj: lrhtraj = false" << std::endl;
204     pobs_.reset(new Observer<MODEL, State_>(obspace_, hop_, yobs_, xx.obsVar(),
205                                             tslot_, subwindows_, hoptlad_));
206   }
207   return pobs_;
```

- L204. Here an object of **class Observer** is constructed, filled with private membres of already constructed classes. A pointer **pobs_** that points to it is released. This pointer will be added to the **pp list.** All these are preparations for all the **Jo** related **postprocessor** that will act in each forecast step.

- **Jump into 1.7.1.5.2.3.2 *Observer.h: Observer***

1.7.1.5.2 *CostFunction.h: linearize*

```
290  // Setup linear model (and trajectory)
291    tlm_.clear();
292    pp.enrollProcessor(new TrajectorySaver<MODEL>(fguess.state()[0], tlmConf, lowres, fguess.modVar(), tlm_));
293
```

- L292. This prepares the **post-processor** that will compute the **trajectory in low resolution.** The **class TrajectorySaver** stores in a private member the **state in low resolution, e.g, the trajectory.** This is called in the code **xlr_. Jumps:**
- **1.7.1.5.2.4 *TrajectorySaver.h: TrajectorySaver (constr)***
- **1.7.1.5.2.4.1 *State.h: State*.** To construct **xlr_!**
- **1.7.1.5.2.4.1.1 *StateIFS.cc: StateIFS***
- **1.7.1.5.2.4.1.1.1 *FieldsIFS.cc: ifs_field_create_f90,*** calling **FIELDS_CREATE.F90**
- **1.7.1.5.2.4.1.1.2 *FieldsIFS.cc: ifs_field_change_resol_f90,*** calling **FIELDS_FP_CHGE_RESOL.F90**

1.7.1.5.2 *CostFunction.h: linearize*

```
294   // Run NL model
295     CtrlVar_ mfguess(fguess);
296     this->runNL(mfguess, pp);
297
```

- L296. Finally, having all the processors defined, we run the NL forecast along the AW, to get **trajectory** and **other processors**. L295 is a copy of the **first guess HR state** at initial time.

- **Jump to 1.7.1.5.2.5 *CostFct3DVar.h: runNL***
- **Jump to 1.7.1.5.2.5.1 *Model.h: forecast***

1.7.1.5.2.5.1 *Model.h: forecast*

```
102   const util::DateTime end(xx.validTime() + len);
103   Log::info() << "Model:forecast: forecast starting: " << xx << std::endl;
104   this->initialize(xx);
105   post.initialize(xx, end, model_->timeResolution());
106   while (xx.validTime() < end) {
107     this->step(xx, maux);
108     post.process(xx);
109   }
110   post.finalize(xx);
111   this->finalize(xx);
112   Log::info() << "Model:forecast: forecast finished: " << xx << std::endl;
113   ASSERT(xx.validTime() == end);
```

- In 3DVAR, len=0... And xx.validTime()=end ☺, so no while loop!
- The while loop shows how it works the postprocess framework on each time stepping

1.7.1.5.2.5.1 *Model.h: forecast*

```
102   const util::DateTime end(xx.validTime() + len);
103   Log::info() << "Model:forecast: forecast starting: " << xx << std::endl;
104   this->initialize(xx);
105   post.initialize(xx, end, model_->timeResolution());
106   while (xx.validTime() < end) {
107     this->step(xx, maux);
108     post.process(xx);
109   }
110   post.finalize(xx);
111   this->finalize(xx);
112   Log::info() << "Model:forecast: forecast finished: " << xx << std::endl;
113   ASSERT(xx.validTime() == end);
```

- L104. Initialize the model Integration given the initial state:
- **Jump to 1.7.1.5.2.5.1.1 *Model.h: initialize***
- **Jump to 1.7.1.5.2.5.1.1.1 *ModelIFS.cc: initialize***
- **Jump to 1.7.1.5.2.5.1.1.1.1 *ifs_prepare_integration_f90,*** which calls **MODEL_INIT.F90** and **MODEL_INI_PHYS.F90**

1.7.1.5.2.5.1 *Model.h: forecast*

```
102   const util::DateTime end(xx.validTime() + len);
103   Log::info() << "Model:forecast: forecast starting: " << xx << std::endl;
104   this->initialize(xx);
105   post.initialize(xx, end, model_->timeResolution());
106   while (xx.validTime() < end) {
107     this->step(xx, maux);
108     post.process(xx);
109   }
110   post.finalize(xx);
111   this->finalize(xx);
112   Log::info() << "Model:forecast: forecast finished: " << xx << std::endl;
113   ASSERT(xx.validTime() == end);
```

- L105. Initialize the postprocessors. **post** is a **list** of pointers each element pointing to a **PostBase object. Method post.initialize** jumps to a intermediate loop on all the processors **in PostProcessor.h**

- **Jump to 1.7.1.5.2.5.1.2 *PostProcessor.h: initialize***

1.7.1.5.2.5.1.2 *PostProcessor.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56                    const util::Duration & step) {
57      BOOST_FOREACH(boost::shared_ptr<PostBase_> jp, processors_)
58        jp->initialize(xx, end, step);
59    }
```

- L57. BOOST_FOREACH is a loop on each element **(jp)** of list **processors_,** in total **2 elements** in our case. The **computation of model at obs locations** (Observer.h) and the **trajectory computation** (TrajectorySaver.h). First model at obs locs, where **xx** is the HR first guess. This to compute departures H(xx) - y...

- **Jump to 1.7.1.5.2.5.1.2.1 *PostBase.h: initialize***

1.7.1.5.2.5.1.2.1 *PostBase.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56        const util::Duration & tstep) {
57      timer_.initialize(xx.validTime(), end, tstep);
58      this->doInitialize(xx, end, tstep);
59      if (timer_.itIsTime(xx.validTime())) this->doProcessing(xx);
60    }
```

- L57. Timming initialization for posprocessors.

- **Jump to 1.7.1.5.2.5.1.2.1.1 *PostTimer.cc: initialize:***
- In this routine we are setting the private members **bgn_** and **end_** (or others if requested in json file) and keeping them in the **class PostTimer** (see PostTimer.h).

1.7.1.5.2.5.1.2.1 *PostBase.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56        const util::Duration & tstep) {
57      timer_.initialize(xx.validTime(), end, tstep);
58      this->doInitialize(xx, end, tstep);
59      if (timer_.itIsTime(xx.validTime())) this->doProcessing(xx);
60    }
```

- L58. Time and geometry initializations for the first posprocessor (Observer.h) done here.

- **Jump to 1.7.1.5.2.5.1.2.1.2 *Observer.h: doInitialize***
- This routine defines **bgn_ ,end_ and hslot_** for the **class Observer.** It seems these will be compared to the ones in class PostTimer...??
- And to get the geometry of obs locations to compute goms

1.7.1.5.2.5.1.2.1.2 *Observer.h: doInitialize*

```
145   if (bgn_ < winbgn_) bgn_ = winbgn_;
146   if (end_ > winend_) end_ = winend_;
147 // Pass the Geometry for IFS -- Bad...
148   gom_.reset(new GOM_(obspace_, hop_.variables(), xx, bgn_, end_));
149   if (setLowResGOM_) gomlr_.reset(new GOM_(obspace_, hop_.variables(), *xxlr_, bgn_, end_));
150 }
```

- L148. We get a pointer **gom_** that points to an object of a **class GOM_** which has the info for the **geometry of obs locations**. **setLowResGOM_** false in our case…
- **1.7.1.5.2.5.1.2.1.2.1 *ModelAtLocations.h: ModelAtLocations***
- **1.7.1.5.2.5.1.2.1.2.1.1 *GomsIFS.h: GomsIFS***
- **1.7.1.5.2.5.1.2.1.2.1.1.1 *GomData.cc: GomData***
- **1.7.1.5.2.5.1.2.1.2.1.1.1.1 *ifs_gom_create_f90,*** that calls **OBS_LOCS_CREATE.F90** and procedure **CREATE** from **SUPERGOM_CLASS.F90**

1.7.1.5.2.5.1.2.1 *PostBase.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56        const util::Duration & tstep) {
57      timer_.initialize(xx.validTime(), end, tstep);
58      this->doInitialize(xx, end, tstep);
59      if (timer_.itIsTime(xx.validTime())) this->doProcessing(xx);
60    }
```

- L59. What is happening is: “if we are at a postprocessing time, then do the Processing”. **timer_.itIsTime(xx.validTime())** returns a boolean. In 3DVAR this is **true.** at analysis time. **Jump to:**

- **1.7.1.5.2.5.1.2.1.3 *Observer.h: doProcessing***
- Here **horizontal interpolation** of state as vertical profiles to obs locations is performed and the result is stored in private member **gom_ of class Observer**

- **1.7.1.5.2.5.1.2.1.3 *Observer.h: doProcessing***

```
159   // Get locations info for interpolator
160     Locations_ locs(obspace_, t1, t2);
161
162   // Interpolate state variables to obs locations
163     xx.interpolate(locs, *gom_);
```

- L160. Jumps:
- **1.7.1.5.2.5.1.2.1.3.1 *Locations.h: Locations***
- **1.7.1.5.2.5.1.2.1.3.1.1 *LocationsIFS.cc: Locations***
- **1.7.1.5.2.5.1.2.1.3.1.1.1 *ifs_obs_locations_f90,*** that calls procedure **CREATE** from **LOCATIONS_MOD.F90**
- L163 Jumps to:
- **1.7.1.5.2.5.1.2.1.3.2 *State.h: interpolate***
- **1.7.1.5.2.5.1.2.1.3.2.1 *StateIFS.cc: interpolate***
- **1.7.1.5.2.5.1.2.1.3.2.1.1 *FieldsIFS.cc: interpolate***
- **1.7.1.5.2.5.1.2.1.3.2.1.1.1 *ifs_field_interp_f90,*** that calls **FIELD_INTERP.F90**

1.7.1.5.2.5.1.2 *PostProcessor.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56                    const util::Duration & step) {
57      BOOST_FOREACH(boost::shared_ptr<PostBase_> jp, processors_)
58        jp->initialize(xx, end, step);
59    }
```

- L57. BOOST_FOREACH is a loop on each element **(jp)** of list **processors_,** in total **2 elements** in our case. The **computation of goms** (Observer.h) and the **trajectory computation** (TrajectorySaver.h). Now trajectory computation, where **xx** is the HR state, some some change of resolution must be involved here...

- **Jump to 1.7.1.5.2.5.1.2.2 *PostBase.h: initialize***

1.7.1.5.2.5.1.2.2 *PostBase.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56        const util::Duration & tstep) {
57      timer_.initialize(xx.validTime(), end, tstep);
58      this->doInitialize(xx, end, tstep);
59      if (timer_.itIsTime(xx.validTime())) this->doProcessing(xx);
60    }
```

- L57. Timming initialization for posprocessors.

- **Jump to 1.7.1.5.2.5.1.2.2.1 *PostTimer.cc: initialize:***
- In this routine we are setting the private members **bgn_** and **end_** (or others if requested in json file) and keeping them in the **class PostTimer** (see PostTimer.h).

1.7.1.5.2.5.1.2.2 *PostBase.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56        const util::Duration & tstep) {
57      timer_.initialize(xx.validTime(), end, tstep);
58      this->doInitialize(xx, end, tstep);
59      if (timer_.itIsTime(xx.validTime())) this->doProcessing(xx);
60    }
```

- L58. Time and geometry initializations for the second postprocessor (TrajectorySaver.h) done here. Jumps:
- **1.7.1.5.2.5.1.2.2.2 *TrajectorySaver.h: doInitialize***
- **1.7.1.5.2.5.1.2.2.2.1 *LinearModel.h: LinearModel***

(Linear Model Factory! See instantiateTlmFactory.h)

- **1.7.1.5.2.5.1.2.2.2.1.1 *LinearModelBase.h: create***
- **1.7.1.5.2.5.1.2.2.2.1.1.1 *LinearModelBase.h: make***
- **1.7.1.5.2.5.1.2.2.2.1.1.1.1 *LinearModelIFS.cc: LinearModelIFS***

- **1.7.1.5.2.5.1.2.2.2.1.1.1.1 *LinearModelIFS.cc: LinearModelIFS***

```
21  // -----------------------------------------------------------------------------
22  LinearModelIFS::LinearModelIFS(const GeometryIFS & resol, const eckit::Configuration & tlConf)
23    : tlm_(0), tstep_(tlConf.getString("tstep")), resol_(resol), traj_(),
24      lrmodel_(resol_, eckit::LocalConfiguration(tlConf, "trajectory"))
25  {
26    const eckit::Configuration * configc = &tlConf;
27    bool linear(1);
28    ifs_setup_f90(tlm_, &configc, resol.toFortran(),linear);  // Same as NL model?
29    Log::info() << "LinearModelIFS created" << std::endl;
30  }
```

- L24. In the constructor list private object **lrmodel_** is defined with the low resolution **resol_**. This lrmodel_ prepare a low resolution NL model setup. Note: trajectory is a NL forecast! Jumps:

- **1.7.1.5.2.5.1.2.2.2.1.1.1.1.1 *ModelIFS.cc: ModelIFS***
- **1.7.1.5.2.5.1.2.2.2.1.1.1.1.1.1 *ifs_setup_f90,*** which eventually calls **MODEL_CREATE.F90**

- **1.7.1.5.2.5.1.2.2.2.1.1.1.1 *LinearModelIFS.cc: LinearModelIFS***

```
21  // -----------------------------------------------------------------------------
22  LinearModelIFS::LinearModelIFS(const GeometryIFS & resol, const eckit::Configuration & tlConf)
23    : tlm_(0), tstep_(tlConf.getString("tstep")), resol_(resol), traj_(),
24      lrmodel_(resol_, eckit::LocalConfiguration(tlConf, "trajectory"))
25  {
26    const eckit::Configuration * configc = &tlConf;
27    bool linear(1);
28    ifs_setup_f90(tlm_, &configc, resol.toFortran(),linear);  // Same as NL model?
29    Log::info() << "LinearModelIFS created" << std::endl;
30  }
```

- L28. Fills object pointed by **tlm_** with setup info. This object **prepares the integration of the linear model** during minimization (correct?). the Jumps:

- **1.7.1.5.2.5.1.2.2.2.1.1.1.1.2 *ifs_setup_f90,*** which eventually calls **MODEL_CREATE.F90** now reading the namelist for the linear model

1.7.1.5.2.5.1.2.2 *PostBase.h: initialize*

```
55    void initialize(FLDS & xx, const util::DateTime & end,
56        const util::Duration & tstep) {
57      timer_.initialize(xx.validTime(), end, tstep);
58      this->doInitialize(xx, end, tstep);
59      if (timer_.itIsTime(xx.validTime())) this->doProcessing(xx);
60    }
```

- L59. What is happening is: “if we are at a postprocessing time, then do the Processing”. **timer_.itIsTime(xx.validTime())** returns a boolean. In 3DVAR this is **true** (although trajectory is not needed…) at analysis time. **Jump to:**
- **1.7.1.5.2.5.1.2.2.3 *TrajectorySaver.h: doProcessing***
- Here **we compute the low resolution NL state having the high resolution NL one**… eventually calling the **MODEL_STEP_TRAJ** F90 routine!

- **1.7.1.5.2.5.1.2.2.3 *TrajectorySaver.h: doProcessing***

```
75  void TrajectorySaver<MODEL>::doProcessing(const State_ & xx) {
76    ASSERT(subtlm_ != 0);
77    subtlm_->setTrajectory(xx, xlr_, lrBias_);
78  }
```

- L77. Jumps:
- **1.7.1.5.2.5.1.2.2.3.1 *LinearModel.h: setTrajectory***
- **1.7.1.5.2.5.1.2.2.3.1.1 *LinearModelBase.h: setTrajectory***
- **1.7.1.5.2.5.1.2.2.3.1.1.1 *LinarModelIFS.cc: setTrajectory***

```
40  void LinearModelIFS::setTrajectory(const StateIFS & xx, StateIFS & xlr, const ModelBias &) {
41    StateIFS xxlr(resol_, xx);
42    //xlr.changeResolution(xx);
43    traj_[xx.validTime()] = lrmodel_.getTrajectory(xxlr, tlm_);
44  }
```

- **1.7.1.5.2.5.1.2.2.3.1.1.1 *LinarModelIFS.cc: setTrajectory***

```
40  void LinearModelIFS::setTrajectory(const StateIFS & xx, StateIFS & xlr, const ModelBias &) {
41    StateIFS xxlr(resol_, xx);
42    //xlr.changeResolution(xx);
43    traj_[xx.validTime()] = lrmodel_.getTrajectory(xxlr, tlm_);
44  }
```

- L41. Construct LR state **xxlr** by interpolating HR xx in GP. Jumps:
- **1.7.1.5.2.5.1.2.2.3.1.1.1.1 *StateIFS.cc: StateIFS***
- **1.7.1.5.2.5.1.2.2.3.1.1.1.1.1 *FieldsIFS.cc: changeResolution***
- **1.7.1.5.2.5.1.2.2.3.1.1.1.1.1.1 *ifs_field_change_resol_f90***, which calls **FIELDS_FP_CHGE_RESOL.F90**
- L43. The idea is to make 1 time step in LR and GP space because **xxlr_** is not an appropriate model state since it is created by interpolation! Proper state trajectory stored in **traj_**
- **1.7.1.5.2.5.1.2.2.3.1.1.1.2 *ModelIFS.cc: getTrajectory***
- **1.7.1.5.2.5.1.2.2.3.1.1.1.2.1 *ifs_prop_traj_f90,*** which eventually calls **MODEL_STEP_TRAJ.F90**

1.7.1.5.2.5.1 *Model.h: forecast*

```
102   const util::DateTime end(xx.validTime() + len);
103   Log::info() << "Model:forecast: forecast starting: " << xx << std::endl;
104   this->initialize(xx);
105   post.initialize(xx, end, model_->timeResolution());
106   while (xx.validTime() < end) {
107     this->step(xx, maux);
108     post.process(xx);
109   }
110   post.finalize(xx);
111   this->finalize(xx);
112   Log::info() << "Model:forecast: forecast finished: " << xx << std::endl;
113   ASSERT(xx.validTime() == end);
```

- L106 – L109. In 3DVAR we don't do this loop since xx.validTime()=end. **Look at it in 4DVAR!** This important look makes the NL HR forecast along the AW and applies the pp if needed in each step.
- L110. So we go to **post.finalize(xx).** It executes **doFinalize** methods for each pp**. Jump to 1.7.1.5.2.5.1.3 *PostProcessor.h: finalize***

1.7.1.5.2.5.1.3 *PostProcessor.h: finalize*

```
69   void finalize(const FLDS & xx) {
70     BOOST_FOREACH(boost::shared_ptr<PostBase_> jp, processors_) jp->finalize(xx);
71   }
```

- BOOST_FOREACH with 2 elements... First pp H(x) related. Jumps:

- **1.7.1.5.2.5.1.3.1 *PostBase.h: finalize***
- **1.7.1.5.2.5.1.3.1.1 *Observer.h: doFinalize***

1.7.1.5.2.5.1.3.1.1 *Observer.h: doFinalize*

```
173 void Observer<MODEL, STATE>::doFinalize(const STATE &) {
174   if (htlad_) {
175     if (setLowResGOM_) {
176       htlad_->setTrajectory(*gomlr_, ybias_);
177     } else {
178       htlad_->setTrajectory(*gom_, ybias_);
179     }
180   }
181   yobs_->runObsOperator(hop_, *gom_, ybias_);
182   gom_.reset();
183   if (setLowResGOM_) gomlr_.reset();
184 }
```

- L178. Here we create the **gom5** and **obsbias5** objects from gom and ybias. **gom** are the model state vertical profiles interpolated at obs locations, and have been computed before **for all AW**.
- In minimization, **gom5** will store direct model fields and **gom** tangent linear and adjoint fields. Jumps:
- **1.7.1.5.2.5.1.3.1.1.1 *LinearObsOperator.h: setTrajectory***
- **1.7.1.5.2.5.1.3.1.1.1.1 *AllObsTLAD: setTrajectory***

1.7.1.5.2.5.1.3.1.1 *Observer.h: doFinalize*

```
173 void Observer<MODEL, STATE>::doFinalize(const STATE &) {
174   if (htlad_) {
175     if (setLowResGOM_) {
176       htlad_->setTrajectory(*gomlr_, ybias_);
177     } else {
178       htlad_->setTrajectory(*gom_, ybias_);
179     }
180   }
181   yobs_->runObsOperator(hop_, *gom_, ybias_);
182   gom_.reset();
183   if (setLowResGOM_) gomlr_.reset();
184 }
```

- L181. Having model at location (goms), **compute H(x) = Observation equivalents** an store them in **hop_.** Jumps:
- **1.7.1.5.2.5.1.3.1.1.2 *Observations.h: runObsOperator***
- **1.7.1.5.2.5.1.3.1.1.2.1 *ObsOperator.h: obsEquiv***
- **1.7.1.5.2.5.1.3.1.1.2.1.1 *AllObs.cc: obsEquiv***
- **1.7.1.5.2.5.1.3.1.1.2.1.1.1 *allobs_equiv_f90 (which call OBS_EQUIV.F90)***

1.7.1.5.2.5.1.3 *PostProcessor.h: finalize*

```
69   void finalize(const FLDS & xx) {
70     BOOST_FOREACH(boost::shared_ptr<PostBase_> jp, processors_) jp->finalize(xx);
71   }
```

- BOOST_FOREACH with 2 elements... Second pp trajectory related. Jumps:

- **1.7.1.5.2.5.1.3.2 *PostBase.h: finalize***
- **1.7.1.5.2.5.1.3.2.1 *TrajectorySaver.h: doFinalize***

```
81 void TrajectorySaver<MODEL>::doFinalize(const State_ &) {
82   tlm_.push_back(subtlm_);
83   subtlm_ = 0;
```

- L82. As we are in the final step, we store the trajectory in this step **subtlm_** in the list of trajectory states along the AW **tlm_**

1.7.1.5.2.5.1 *Model.h: forecast*

```
102   const util::DateTime end(xx.validTime() + len);
103   Log::info() << "Model:forecast: forecast starting: " << xx << std::endl;
104   this->initialize(xx);
105   post.initialize(xx, end, model_->timeResolution());
106   while (xx.validTime() < end) {
107     this->step(xx, maux);
108     post.process(xx);
109   }
110   post.finalize(xx);
111   this->finalize(xx);
112   Log::info() << "Model:forecast: forecast finished: " << xx << std::endl;
113   ASSERT(xx.validTime() == end);
```

- L111. It does nothing but indicate model forecast has reached the end, calling some intemediate time util rutines and giving some debug info. Jumps:

- **1.7.1.5.2.5.1.4 *Model.h: finalize***
- **1.7.1.5.2.5.1.4.1 *ModelIFS.cc: finalize***

1.7.1.5.2 *CostFunction.h: linearize*

```
298  // Cost function value
299    costJb_ = jb_->finalizeTraj(jq, innerConf);
300    costJoJc_ = 0.0;
301    for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
302      costJoJc_ += jterms_[jj].finalizeTraj(diagnostic);
303    }
304    double costJ = costJb_ + costJoJc_;
305    Log::test() << "CostFunction: Nonlinear J = " << costJ << std::endl;
306    return costJ;
307  }
```

- L296-L304. This block computes the **scalar value of CostJ = costJb + CostJo.** For diagnostic purposes only??

$$J(\delta\mathbf{x}) = \frac{1}{2}\|\delta\mathbf{x}\|^2_{\mathbf{B}^{-1}} + \frac{1}{2}\|\mathbf{H}\delta\mathbf{x} - \mathbf{d}\|^2_{\mathbf{R}^{-1}}$$

$$J(\delta\mathbf{x}) = \frac{1}{2}\delta\mathbf{x}^{\mathrm{T}}\mathbf{B}^{-1}\delta\mathbf{x} + \frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i) - \mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i) - \mathbf{d}_i)$$

- L299. Computation **of CostJb scalar value,** Jump to **1.7.1.5.2.6 *CostJbTotal.h: finalizeTraj***

1.7.1.5.2.6 *CostJbTotal.h: finalizeTraj*

```
204  // Compute x_0 - x_b for Jb
205    jb_->computeIncrement(xb_.state(), fg_->state(), dxFG_->state());
206
```

- This line computes **first guess increment,** difference of background and first guess between outer loops. In first outer iteration this should be 0… **dx** in equations (slide before). Jumps:
- **1.7.1.5.2.6.1 *CostJb3D.h: computeIncrement***
- **1.7.1.5.2.6.1.1 *Increment4D.h: diff***
- **1.7.1.5.2.6.1.1.1 *Increment.h: diff***
- **1.7.1.5.2.6.1.1.1.1 *IncrementIFS.cc: diff***
- **1.7.1.5.2.6.1.1.1.1.1 *FieldsIFS.cc: diff***
- **1.7.1.5.2.6.1.1.1.1.1.1 *ifs_field_diff_incr_f90,*** which calls **FIELDS_DIFF_INCR.F90**

1.7.1.5.2.6 *CostJbTotal.h: finalizeTraj*

```
207   // Model and Obs biases
208     dxFG_->modVar().diff(fg_->modVar(), xb_.modVar());
209     dxFG_->obsVar().diff(fg_->obsVar(), xb_.obsVar());
210
```

- Same for **model error** and **varbc bias** terms of control vector
- L208. Model error not implemented in CY46. Jumps
- **1.7.1.5.2.6.2 *ModelAuxIncrement.h: diff***
- **1.7.1.5.2.6.2.1 *ModelBiasIncrement.h: diff (empty function)***

- L209. Varbc part. Jumps:
- **1.7.1.5.2.6.3 *ObsAuxIncrement.h: diff***
- **1.7.1.5.2.6.3.1 *ObsBiasIncrement.cc: diff***
- **1.7.1.5.2.6.3.1.1 *ifs_obs_bias_increment_diff_f90***
- **1.7.1.5.2.6.3.1.2 *obsbias_tcv_increment_diff_f90***

1.7.1.5.2.6 *CostJbTotal.h: finalizeTraj*

```
227  // Return Jb value
228    double zjb = this->evaluate(*dxFG_);
229    return zjb;
```

$$J(\delta \mathbf{x}) = \frac{1}{2}\delta \mathbf{x}^{\mathrm{T}} \mathbf{B}^{-1} \delta \mathbf{x}$$

- Computation of CostJb scalar value having **dx**. Jump:

- **1.7.1.5.2.6.4 *CostJbTotal.h: evaluate***

```
235  double CostJbTotal<MODEL>::evaluate(const CtrlInc_ & dx) const {
236    CtrlInc_ gg(*this);
237    this->multiplyBinv(dx, gg);
```

- **1.7.1.5.2.6.4.1 *CostJbTotal.h: multiplyBinv***

1.7.1.5.2.6.4.1 *CostJbTotal.h: multiplyBinv*

```
334 void CostJbTotal<MODEL>::multiplyBinv(const CtrlInc_ & dxin, CtrlInc_ & dxout) const {
335   jb_->Bminv(dxin.state(), dxout.state());
336   jbModBias_.inverseMultiply(dxin.modVar(), dxout.modVar());
337   jbObsBias_.inverseMultiply(dxin.obsVar(), dxout.obsVar());
338 }
```

- L335 jumps:
- **1.7.1.5.2.6.4.1.1 *CostJb3D.h: Bminv***
- **1.7.1.5.2.6.4.1.1.1 *ErrorCovariance.h: inverseMultiply***
- **1.7.1.5.2.6.4.1.1.1.1 *ErrorCovariance3D.cc: inverseMultiply***
- **1.7.1.5.2.6.4.1.1.1.1.1 *ifs_error_covariance_3d_invmult_f90***

1.7.1.5.2.6.4.1 *CostJbTotal.h: multiplyBinv*

```
334 void CostJbTotal<MODEL>::multiplyBinv(const CtrlInc_ & dxin, CtrlInc_ & dxout) const {
335   jb_->Bminv(dxin.state(), dxout.state());
336   jbModBias_.inverseMultiply(dxin.modVar(), dxout.modVar());
337   jbObsBias_.inverseMultiply(dxin.obsVar(), dxout.obsVar());
338 }
```

- L336 jumps:
- **1.7.1.5.2.6.4.1.2 *ModelAuxCovariance.h: inverseMultiply***
- **1.7.1.5.2.6.4.1.2.1 *ModelBiasCovariance.h: inverseMultiply***
- L337 jumps:
- **1.7.1.5.2.6.4.1.3 *ObsAuxCovariance.h: inverseMultiply***
- **1.7.1.5.2.6.4.1.3.1 *ObsBiasCovariance.cc: inverseMultiply***
- **1.7.1.5.2.6.4.1.3.1.1 *ifs_obs_bias_covariance_invmult_f90,*** calls **INVMULT.F90**
- **1.7.1.5.2.6.4.1.3.1.2 *obsbias_tcv_covariance_invmult_f90,*** calls **INVMULT_TOVSCV_BGC.F90**

1.7.1.5.2.6 *CostJbTotal.h: finalizeTraj*

```
227  // Return Jb value
228    double zjb = this->evaluate(*dxFG_);
229    return zjb;
```

$$J(\delta \mathbf{x}) = \frac{1}{2}\delta \mathbf{x}^{\mathrm{T}}\mathbf{B}^{-1}\delta \mathbf{x}$$

- Computation of CostJb scalar value having **dx**. Jump:

- **1.7.1.5.2.6.4 *CostJbTotal.h: evaluate***

```
239    double zjb = 0.0;
240    double zz = 0.5 * dot_product(dx.state(), gg.state());
241    Log::info() << "CostJb   : Nonlinear Jb State = " << zz << std::endl;
242    zjb += zz;
243    zz = 0.5 * dot_product(dx.modVar(), gg.modVar());
244    Log::info() << "CostJb   : Nonlinear Jb Model Aux = " << zz << std::endl;
245    zjb += zz;
246    zz = 0.5 * dot_product(dx.obsVar(), gg.obsVar());
247    Log::info() << "CostJb   : Nonlinear Jb Obs Aux = " << zz << std::endl;
248    zjb += zz;
249
250    Log::info() << "CostJb   : Nonlinear Jb = " << zjb << std::endl;
```

- **dot_product** is the C++ function that computes the first part of LHS of $J(\delta \mathbf{x}) = \frac{1}{2}\delta \mathbf{x}^{\mathrm{T}}\mathbf{B}^{-1}\delta \mathbf{x}$. So we get CostJb for the 3 terms and total.

1.7.1.5.2 *CostFunction.h: linearize*

```
298  // Cost function value
299    costJb_ = jb_->finalizeTraj(jq, innerConf);
300    costJoJc_ = 0.0;
301    for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
302      costJoJc_ += jterms_[jj].finalizeTraj(diagnostic);
303    }
304    double costJ = costJb_ + costJoJc_;
305    Log::test() << "CostFunction: Nonlinear J = " << costJ << std::endl;
306    return costJ;
307  }
```

- L296-L304. This block computes the **scalar value of CostJ = costJb + CostJo.** For diagnostic purposes only??

$$J(\delta\mathbf{x}) = \frac{1}{2}\|\delta\mathbf{x}\|^2_{\mathbf{B}^{-1}} + \frac{1}{2}\|\mathbf{H}\delta\mathbf{x} - \mathbf{d}\|^2_{\mathbf{R}^{-1}}$$

$$J(\delta\mathbf{x}) = \frac{1}{2}\delta\mathbf{x}^{\mathrm{T}}\mathbf{B}^{-1}\delta\mathbf{x} + \frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i) - \mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i) - \mathbf{d}_i)$$

- L302. Computation **of CostJo scalar value,** Jump to **1.7.1.5.2.7 *CostJo.h: finalizeTraj***

1.7.1.5.2.7 *CostJo.h: finalizeTraj*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$

```
216   boost::scoped_ptr<Observations_> yeqv(pobs_->release());
217   Log::info() << "Jo Observation Equivalent:" << *yeqv << std::endl;
218   R_.linearize(*yeqv);
```

- L218. What is done here? The last F90 says it is nothing implemented in Cy46... **yeqv has precomputed H(x).** Jumps:
- **1.7.1.5.2.7.1 *ObsErrorCovariance.h: linearize***
- **1.7.1.5.2.7.1.1 *ObsErrorBase.h: linearize***
- **1.7.1.5.2.7.1.1.1 *AllObsCovariance.cc: linearize***
- **1.7.1.5.2.7.1.1.1.1 *ifs_r_linearize_f90 (which calls R_LINEARIZE.F90 which has nothing implemented)***

1.7.1.5.2.7 *CostJo.h: finalizeTraj*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$ $\mathbf{d}_i=\mathbf{y}_i^{\mathrm{o}}-H_i\mathbf{x}^{\mathrm{b}}(t_i)$

```
220   Departures_ ydep(*yeqv - yobs_);
221   Log::info() << "Jo Departures:" << ydep << std::endl;
222
223   Log::info() << "Warning Departures forced by: ombg";
224   Departures_ forcing_dep(ydep);
225   forcing_dep.read("fg_depar@body");
226   forcing_dep *= -1.0;
227   ydep *= 0.0;
228   ydep += forcing_dep;
```

- L220. Compute **-departures.** Jumps:
- **1.7.1.5.2.7.2 *Observations.h: operator-***
- **1.7.1.5.2.7.2.1 *ObsVector.h: ObsVector***
- **1.7.1.5.2.7.2.1.1 *ObsVector.cc: ObsVector***
- **1.7.1.5.2.7.2.1.1.1 *ifs_obsvec_clone_f90***

1.7.1.5.2.7 *CostJo.h: finalizeTraj*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$ $\mathbf{d}_i=\mathbf{y}_i^{\mathrm{o}}-H_i\mathbf{x}^{\mathrm{b}}(t_i)$

```
220   Departures_ ydep(*yeqv - yobs_);
221   Log::info() << "Jo Departures:" << ydep << std::endl;
222
223   Log::info() << "Warning Departures forced by: ombg";
224   Departures_ forcing_dep(ydep);
225   forcing_dep.read("fg_depar@body");
226   forcing_dep *= -1.0;
227   ydep *= 0.0;
228   ydep += forcing_dep;
```

- L224. Clones precomputed departures
- **1.7.1.5.2.7.3 *Departures.h: Departures***
- **1.7.1.5.2.7.3.1 *ObsVector.h: ObsVector***
- **1.7.1.5.2.7.3.1.1 *ObsVector.cc: ObsVector***
- **1.7.1.5.2.7.3.1.1.1 *ifs_obsvec_clone_f90***

1.7.1.5.2.7 *CostJo.h: finalizeTraj*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$ $\mathbf{d}_i=\mathbf{y}_i^{\mathrm{o}}-H_i\mathbf{x}^{\mathrm{b}}(t_i)$

```
220   Departures_ ydep(*yeqv - yobs_);
221   Log::info() << "Jo Departures:" << ydep << std::endl;
222
223   Log::info() << "Warning Departures forced by: ombg";
224   Departures_ forcing_dep(ydep);
225   forcing_dep.read("fg_depar@body");
226   forcing_dep *= -1.0;
227   ydep *= 0.0;
228   ydep += forcing_dep;
```

- L225. Reads and uses **precomputed ODB departures**. Jumps:
- **1.7.1.5.2.7.4 *Departures.h: read***
- **1.7.1.5.2.7.4.1 *ObsVector.h: read***
- **1.7.1.5.2.7.4.1.1 *ObsVector.cc: read***
- **1.7.1.5.2.7.4.1.1.1 *ifs_obsvec_read_f90***

1.7.1.5.2.7 *CostJo.h: finalizeTraj*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$ $\mathbf{d}_i=\mathbf{y}_i^{\mathrm{o}}-H_i\mathbf{x}^{\mathrm{b}}(t_i)$

```
230    gradFG_.reset(R_.inverseMultiply(ydep));
231
232    double zjo = this->printJo(ydep, *gradFG_);
```

- L230. **Compute R^-1(departures),** RHS of CostJo. **Not clear, according to equation in ( ) there are no depart but yobs...??**
- **1.7.1.5.2.7.5 *ObsErrorCovariance.h: inverseMultiply***
- **1.7.1.5.2.7.5.1 *ObsErrorBase.h: inverseMultiply***
- **1.7.1.5.2.7.5.1.1 *AllObsCovariance.cc: inverseMultiply***
- **1.7.1.5.2.7.5.1.1.1 *ObsVector.cc: ObsVector***
- **1.7.1.5.2.7.5.1.1.1.1 *ifs_obsvec_clone_f90***
- **1.7.1.5.2.7.5.1.1.2 *ifs_r_imul_f90***

1.7.1.5.2.7 *CostJo.h: finalizeTraj*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$ $\mathbf{d}_i=\mathbf{y}_i^{\mathrm{o}}-H_i\mathbf{x}^{\mathrm{b}}(t_i)$

```
231
232     double zjo = this->printJo(ydep, *gradFG_);
233
```

- L232. **Compute CostJo and print debug info,** LHS of Jo:
- **1.7.1.5.2.7.6 *CostJo.h: printJo:***

```
307   const double zjo = 0.5 * dot_product(dy, grad);
308
309   // print Jo table
310   obspace_.printJo(dy, grad);
311
312   // print total Jo
313   const unsigned nobs = dy.numberOfObs();
314   if (nobs > 0) {
315     Log::test() << "CostJo   : Nonlinear Jo = " << zjo
316                 << ", nobs = " << nobs << ", Jo/n = " << zjo/nobs
317                 << ", err = " << R_.getRMSE() << std::endl;
```

1.7.1.5.2.7.6 *CostJo.h: printJo*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$ $\mathbf{d}_i=\mathbf{y}_i^{\mathrm{o}}-H_i\mathbf{x}^{\mathrm{b}}(t_i)$

```
307   const double zjo = 0.5 * dot_product(dy, grad);
308
309   // print Jo table
310   obspace_.printJo(dy, grad);
311
```

- L307. **Compute CostJo** using dot_product...

- L310. Print debug Jo... Jumps:
- **1.7.1.5.2.7.6.1 *ObservationSpace.h: printJo***
- **1.7.1.5.2.7.6.1.1 *ObsSpaceODB.cc: printJo***
- **1.7.1.5.2.7.6.1.1.1 *obs_printjo_f90***

1.7.1.5.2.7.6 *CostJo.h: printJo*

- Computation of: $\frac{1}{2}\sum_{i=0}^{n}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)^{\mathrm{T}}\mathbf{R}_i^{-1}(\mathbf{H}_i\delta\mathbf{x}(t_i)-\mathbf{d}_i)$ $\mathbf{d}_i=\mathbf{y}_i^{\mathrm{o}}-H_i\mathbf{x}^{\mathrm{b}}(t_i)$

```
312   // print total Jo
313   const unsigned nobs = dy.numberOfObs();
314   if (nobs > 0) {
315     Log::test() << "CostJo   : Nonlinear Jo = " << zjo
316                 << ", nobs = " << nobs << ", Jo/n = " << zjo/nobs
317                 << ", err = " << R_.getRMSE() << std::endl;
```

- L315. Print more basic debug info on Jo
- L317. To get the RMSE of R... **Mean of all diagonal elements?**
- **1.7.1.5.2.7.6.2 *ObsErrorCovariance.h: getRMSE***
- **1.7.1.5.2.7.6.2.1 *AllObsCovariance.cc: getRMSE***
- **1.7.1.5.2.7.6.2.1 *ifs_r_rmse_f90***

- Finalize J.linearize!

```
69  //  Setup quadratic problem
70      J.linearize(xx, iterconfs[jouter], post);
71
```

1.7.1.5 IncrementalAssimilation.h: *IncrementalAssimilation()*

```
72  //  Minimization
73      boost::scoped_ptr<CtrlInc_> dx(minim->minimize(iterconfs[jouter]));
74
```

- Main jobs here are 2: **solve the linear system $\nabla J(X)=0$** where **X is the control vector** and **apply several tests of stability.** The output is the **analysis increment in physical space, dx.**

- **Jump to 1.7.1.5.3 *Minimizer.h: minimize***

1.7.1.5.3 *Minimizer.h: minimize*

```
74  ControlIncrement<MODEL> * Minimizer<MODEL>::minimize(const eckit::Configuration & config) {
75    // TLM tests
76    this->tlmTests(config);
77
78    // ADJ tests
79    this->adjTests(config);
80
81    // Minimize
82    ControlIncrement<MODEL> * dx = this->doMinimize(config);
83
84    // TLM propagation test
85    this->tlmPropagTest(config, *dx);
86
87    // Update outer loop counter
88    outerIteration_++;
89
90    return dx;
91  }
```

- **So far all the tests are left out of this documentation. Need to understand and document them!**

- **Jump to 1.7.1.5.3.1 *SQRTMinimizer.h: doMinimize***

1.7.1.5.3.1 *SQRTMinimizer.h: doMinimize*

```
119     Jb.test(utrhs);
120     Jb.updateGradientFG(*dxsum_, *gradJb_);
121     Log::info() << "DEBUG VARBC: <gradJb_varbc,gradJb_varbc> after updating: " << util::full_precision(dot_product(g
122     Log::info() << "DEBUG VARBC: <utrhs_varbc,utrhs_varbc> before adding gradJb_varbc: " << util::full_precision(dot
123     Log::info() << "DEBUG VARBC: <utrhs_state,utrhs_state> before adding gradJb_state: " << util::full_precision(dot
124     J_.jb().addGradientFG(utrhs, *gradJb_);
125     Log::info() << "DEBUG VARBC: <utrhs_varbc,utrhs_varbc> after adding gradJb_varbc: " << util::full_precision(dot_
126     Log::info() << "DEBUG VARBC: <utrhs_state,utrhs_state> after adding gradJb_state: " << util::full_precision(dot_
127     utrhs *= -1.0;
128     Log::info() << "DEBUG VARBC: <utrhs_varbc,utrhs_varbc> after multiplying by -1: " << util::full_precision(dot_pr
129
130     Log::info() << classname() << " Ut rhs" << utrhs << std::endl;
131
132   // Define minimisation starting point
133     // dv = U dx
134     CtrlVec_ dv(J_.jb());
135
136   // Set J[0] = 0.5 (x_i - x_b)^T B^{-1} (x_i - x_b) + 0.5 d^T R^{-1} d
137     const double costJ0Jb = costJ0Jb_;
138     const double costJ0JbVarBC = costJ0JbVarBC_;
139     const double costJ0JoJc = J_.getCostJoJc();
140
141     Log::info() << "DEBUG VARBC: <dv_varbc,dv_varbc> before minimization: " << util::full_precision(dot_product(dv.o
142
143   // Solve the linear system
144     double reduc = this->solve(dv, utrhs, Jb, UtHtRinvHU, costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc);
145
```

1.7.1.5.3.1 *SQRTMinimizer.h: doMinimize*

- The process consist in **preparing** the inputs for the iterative (inner loop) **linear solver Ax=b,** where in our case: **$\nabla J(X)=0$.** Gradient with respect to **X.** The solution of the solver is the **X** that makes the gradient be (very close to) 0. In our case the **X** is the **control vector**. There are different options for the linear solver, **lanczos, congrad…** and others available in the OOPS code.

- **See Roel's Stapper presentation budapest2019stappers.pdf** for a clear introduction to 4DVAR algebra minimization problem.... **Thanks Roel**! Basically our **solver** solves the equation:

$$(\mathbf{I}+\mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{U})\chi=\mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

$$\mathbf{B}=\mathbf{U}\mathbf{U}^T$$

$$\delta\mathbf{x}=\mathbf{U}\chi$$

- Where **U** is the **square root of B**, and **physical space increment dx** and **control vector X** relates through **U.**

1.7.1.5.3.1 *SQRTMinimizer.h: doMinimize*

```
143  // Solve the linear system
144    double reduc = this->solve(dv, utrhs, Jb, UtHtRinvHU, costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc);
145
```

- L144. This will call the mathematical solver of the linear System. Let's see where each input argument comes from.

- **dv:** control vector **X** at start of minimization. Its value is constructed from precomputed Jb. **ControlVector class** has 2 members in the private section, a **state increment and an observation bias.** See ControlVector.h. It responds to the fact **Obs Bias is part of the J**:

$$J(\mathbf{x},\beta) = \underbrace{\frac{1}{2}\|\mathbf{x}-\mathbf{x}^b\|^2_{\mathbf{B}^{-1}}}_{J_b} + \underbrace{\frac{1}{2}\|\beta-\beta^b\|^2_{\mathbf{B}_\beta{}^{-1}}}_{J_p} + \underbrace{\frac{1}{2}\|\mathcal{H}(\mathbf{x})+\mathbf{P}\beta-\mathbf{y}\|^2_{\mathbf{R}^{-1}}}_{J_o}$$

```
86   private:
87    void print(std::ostream &) const;
88    IncrCtlVec4D_   incrCtlVec4D_;
89    //ModelAuxCtlVec_ modBiasCtlVec_;
90    ObsAuxCtlVec_   obsBiasCtlVec_;
91  };
```

1.7.1.5.3.1 *SQRTMinimizer.h: doMinimize*

```
143  // Solve the linear system
144    double reduc = this->solve(dv, utrhs, Jb, UtHtRinvHU, costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc);
145
```

- **utrhs:** This is the Right-Hand Side of the equation:

$$(\mathbf{I} + \mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{U})\chi = \mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

- To get it first we compute **rhs:** $\mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$. This can be thought as **an state increment** at the beginning of the analysis window computed by **running the adjoint model** (H^T or M^T H^T in 4DVAR) applied to **obs departures.** It is a guess of the state increment at the beginning of the window caused by the observations

1.7.1.5.3.1 *SQRTMinimizer.h: doMinimize*

```
110   // Compute RHS (sum v_{i}) + U^T H^T R^{-1} d
111     CtrlInc_ rhs(J_.jb());
112     J_.computeGradientFG(rhs);
```

- To get **rhs: 1.7.1.5.3.1.1 *CostFunction.h: computeGradient***

```
312 void CostFunction<MODEL>::computeGradientFG(CtrlInc_ & grad) const {
313   PostProcessor<Increment_> pp;
314   PostProcessorAD<Increment_> costad;
315   this->zeroAD(grad);
316
317   for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
318     boost::shared_ptr<const GeneralizedDepartures> tmp(jterms_[jj].newGradientFG());
319     costad.enrollProcessor(jterms_[jj].setupAD(tmp, grad));
320   }
321
322   this->runADJ(grad, costad, pp);
323 }
```

- L313-L320: **grad=0**, construct adjoint **costad pp** that contains **departures**, **obs bias** and **linear adjoint H^T**, all precomputed
- L322: get forecast **grad. 1.7.1.5.3.1.1 *CostFct3DVar.h: runADJ***

1.7.1.5.3.1.1 *CostFct3DVar.h: runADJ*

- Jump to **1.7.1.5.3.1.1.1** ***LinearModel.h: forecastAD***

```
179    Log::info() << "LinearModel<MODEL>::forecastAD: Starting " << dx << std::endl;
180    this->initializeAD(dx);
181    post.initialize(dx, bgn, tstep);
182    cost.initializeAD(dx, bgn, tstep);
```

- L180. Given **dx=0** as input argument. Jumps
- **1.7.1.5.3.1.1.1.1** ***LinearModel.h: initializeAD***
- **1.7.1.5.3.1.1.1.1.1** ***LinearModelBase.h: initializeAD***
- **1.7.1.5.3.1.1.1.1.1.1** ***LinearModelIFS.cc*: *initializeAD***
- **1.7.1.5.3.1.1.1.1.1.1.1** ***ifs_prepare_integration_ad_f90***

1.7.1.5.3.1.1 *CostFct3DVar.h: runADJ*

- Jump to **1.7.1.5.3.1.1.1 *LinearModel.h: forecastAD***

```
179   Log::info() << "LinearModel<MODEL>::forecastAD: Starting " << dx << std::endl;
180   this->initializeAD(dx);
181   post.initialize(dx, bgn, tstep);
182   cost.initializeAD(dx, bgn, tstep);
```

- L181. Jumps to nowhere since **post pp** has no elements in list

- L182. **cost pp** has 1 element in list, apply H^T. Jumps:
- **1.7.1.5.3.1.1.1.2 *PostProcessorAD.h: initializeAD***
- **1.7.1.5.3.1.1.1.2.1 *PostBaseAD.h: initializeAD***
- **1.7.1.5.3.1.1.1.2.1.1 *ObserverAD.h: doFirstAD***

1.7.1.5.3.1.1.1.2.1.1 *ObserverAD.h: doFirstAD*

```
102     if (bgn_ < winbgn_) bgn_ = winbgn_;
103     if (end_ > winend_) end_ = winend_;
104     gom_.reset(new GOM_(obspace_, hoptlad_.variables(), dx, bgn_, end_));
105     ydep_->runObsOperatorAD(hoptlad_, *gom_, ybias_);
106   }
```

- L104. Create **gom columns** that will store H^T!. Jumps:
- **1.7.1.5.3.1.1.1.2.1.1.1 *ModelAtLocations.h: ModelAtLocations***
- **1.7.1.5.3.1.1.1.2.1.1.1.1 *GomsIFS.h: GomsIFS***
- **1.7.1.5.3.1.1.1.2.1.1.1.1.1 *GomData.cc: GomData***
- **1.7.1.5.3.1.1.1.2.1.1.1.1.1.1 *ifs_gom_create_f90***
- L105. Apply H^T to fill **goms** with adjoint data.
- **1.7.1.5.3.1.1.1.2.1.1.2 *Departures.h: runObsOperatorAD***
- **1.7.1.5.3.1.1.1.2.1.1.2.1 *LinearObsOperator.h: obsEquivAD***
- **1.7.1.5.3.1.1.1.2.1.1.2.1.1 *AllObsTLAD.cc: obsEquivAD***
- **1.7.1.5.3.1.1.1.2.1.1.2.1.1.1 *allobs_equiv_ad_f90***

1.7.1.5.3.1.1 *CostFct3DVar.h: runADJ*

- Jump to **1.7.1.5.3.1.1.1** ***LinearModel.h: forecastAD***

```
190     while (dx.validTime() > bgn) {
191       cost.processAD(dx);
192       this->stepAD(dx, mctl);
193       post.process(dx);
194     }
195   }
196   cost.finalizeAD(dx);
197   post.finalize(dx);
198   this->finalizeAD(dx);
```

- L190-194. Nothing done in 3DVAR, it does in 4DVAR…
- L196…

- **In SQRTMinimizer.h :**

```
143  // Solve the linear system
144    double reduc = this->solve(dv, utrhs, Jb, UtHtRinvHU, costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc);
145
```

  - **utrhs:** From Roel's presentation budapest2019stapper, clearly this is the Right-Hand Side of the equation:

$$(\mathbf{I} + \mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{U})\chi = \mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

  - In the C++ code, **uthrs** is computed by calling the **multiplySqrtTrans** method, which eventually calls a F90 routine: **ifs_error_covariance_3d_mult_sqrt_trans_c…**

```
115    CtrlVec_ utrhs(J_.jb());
116    B.multiplySqrtTrans(rhs, utrhs);
```

rhs object?

Next slide…

- **In SQRTMinimizer.h :**

```
143  // Solve the linear system
144    double reduc = this->solve(dv, utrhs, Jb, UtHtRinvHU, costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc);
145
```

  - **rhs** is : $H^T R^{-1} d$

  - And this can be thought as **an state increment** **at the beginning of the analisis window** which is computed by **running the adjoint model** (H^T or M^T H^T in 4DVAR) applied to **obs departures.** It is like a guess of the first state increment at beggining of an window caused by observations

```
110  // Compute RHS (sum v_{i}) + U^T H^T R^{-1} d
111    CtrlInc_ rhs(J_.jb());
112    J_.computeGradientFG(rhs);
```

This **computeGradient** eventually calls several F90 routines, in paricular **ifs_propagate_ad_f90.** Example of adjoint, next slide...

[redacted]

- **In SQRTMinimizer.h :**

Adjoint code an example

- Suppose in nonlinear model we have the statement: $x = y + z^2$
- Corresponding line in tangent linear code: $\delta x = \delta y + 2z\delta z$
- Write as a matrix

$$\begin{bmatrix} \delta z \\ \delta y \\ \delta x \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2z & 1 & 0 \end{bmatrix} \begin{bmatrix} \delta z \\ \delta y \\ \delta x \end{bmatrix}$$

- In the adjoint code corresponding statement is

$$\begin{bmatrix} \delta z^* \\ \delta y^* \\ \delta x^* \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2z \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \delta z^* \\ \delta y^* \\ \delta x^* \end{bmatrix}$$

- i.e

$$\delta z^* = \delta z^* + 2z\delta x^*$$
$$\delta y^* = \delta y^* + \delta x^*$$
$$\delta x^* = 0$$

- **In SQRTMinimizer.h :**

```
143  // Solve the linear system
144    double reduc = this->solve(dv, utrhs, Jb, UtHtRinvHU, costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc);
145
```

  - **Jb:** This is Jb, first term of J, with a given intial value…
  - **UtHtRinvHU:** Clear! **Left Hand Side** of equation to solve…:

$$(\mathbf{I} + \mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}\mathbf{U})\chi = \mathbf{U}^T\mathbf{H}^T\mathbf{R}^{-1}\mathbf{d}$$

  - **costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc** are floats and integers representing the value of distinct terms of cost function J, number of inner loop iterations and norm reduction of J gradient to consider iterative convergence (correct?)

- **In SQRTMinimizer.h :**

```
143  // Solve the linear system
144    double reduc = this->solve(dv, utrhs, Jb, UtHtRinvHU, costJ0Jb, costJ0JbVarBC, costJ0JoJc, ninner, gnreduc);
145
```

  - **dv: (**Updated) output of **solve,** transformed to **dx:**

```
148    CtrlInc_ * dx = new CtrlInc_(J_.jb());
149    B.multiplySqrt(dv, *dx);
```

  - This **multiplySqrt** method eventually calls the F90 routine **ifs_error_covariance_3d_mult_sqrt_c**

- **In SQRTMinimizer.h :**
  - Finally in the code there are **updates of variables** in case we are in a 4DVAR outer loop where nouter>1, we will add this variables to the initial values for the same variables in the next iteration…

```
157   // Update gradient Jb
158     *gradJb_ += dv;
```

```
151     *dxsum += *dx;
```

```
165 // Update Jb component of J[0]: 0.5 (x_i - x_b)^T B^-1 (x_i - x_b)
166   costJ0Jb_ += 0.5 * dot_product(dv, jbdv);
167   costJ0JbVarBC_ += 0.5 * dot_product(dv.obsVar(), jbdv.obsVar());
168   for (unsigned int jouter = 1; jouter < dvh_.size(); ++jouter) {
169     CtrlVec_ dvhtmp(J_.jb(), dvh_[jouter-1]);
170     costJ0Jb_ += dot_product(dv, dvhtmp);
171     costJ0JbVarBC_ += dot_product(dv.obsVar(), dvhtmp.obsVar());
172   }
```

  - Next iteration (iter=2):

```
120   Jb.updateGradientFG(*dxsum_, *gradJb_);
121   Log::info() << "DEBUG VARBC: <gradJb va
122   Log::info() << "DEBUG VARBC: <utrhs_var
123   Log::info() << "DEBUG VARBC: <utrhs_sta
124   J_.jb().addGradientFG(utrhs, *gradJb_);
```

dxsum stores the sum of increments of diferent outer iterations and is stored in the private section of class SQRTMinimizer…

- **In Minimizer.h :**

```
74  ControlIncrement<MODEL> * Minimizer<MODEL>::minimize(const eckit::Configuration & config) {
75    // TLM tests
76    this->tlmTests(config);
77
78    // ADJ tests
79    this->adjTests(config);
80
81    // Minimize
82    ControlIncrement<MODEL> * dx = this->doMinimize(config);
83
84    // TLM propagation test
85    this->tlmPropagTest(config, *dx);
86
87    // Update outer loop counter
88    outerIteration_++;
89
90    return dx;
91  }
```

  - Having finished **doMinimize** method, the big work is done. Now in Minimizer.h there is another test, the **tlmPropagTest.** It would be good to understand this differents tests (next slide from Roel presentation)

[redacted]

- **In Minimizer.h :**

Adjoint test and gradient test

- By definition the adjoint

$$(\mathbf{H}\delta\mathbf{x}, \delta\mathbf{y}) = \left\langle \delta\mathbf{x}, \mathbf{H}^T \delta\mathbf{y} \right\rangle$$

  Validity of the adjoint can be test by computing the left and right hand side for some δx and δy. Equality should hold up to machine precision.

- Gradient test

$$\lim_{h \to 0} \frac{J(\chi + h\delta\chi) - J(\chi)}{\langle \nabla J, h\delta\chi \rangle}$$

  ratio should approach 1 for small enough values of h (but not too small because of round-off errors)
  See NAMVARTEST logicals LADTEST LGRTEST.

- **In IncrementalAssimilation.h :**

```
75  //  Compute analysis in physical space
76      J.addIncrement(xx, *dx);
77
78  //  Reproduce oman in masterodb
79      if (config.has("mimic_masterodb_oman")) {
80        Log::info() << "IncrementalAssimilation: Write analysis ";
81        xx.state().write(eckit::LocalConfiguration(config, "output"));
82
83        Log::info() << "IncrementalAssimilation: Mimic an_depar in var/CNT0";
84        Dual_ vv;
85        const HMatrix<MODEL> H(J);
86        H.multiply(*dx, vv);
87        const Departures_ & dy = dynamic_cast<const Departures_ &>(*vv.getv(0));
88        Departures_ dy0(dy);
89
90        dy0.read("fg_depar@body");
91        dy0 -= dy;
92        dy0.save("an_depar@body");
93
94        const CostJo_ & Jo = dynamic_cast<const CostJo_ &>(J.jterm(0));
95        boost::scoped_ptr < Departures_ > grad(Jo.multiplyCoInv(dy0));
96        double zjo = Jo.printJo(dy0, *grad);
97      }
98
99  //  Clean-up trajectory, etc...
100     J.resetLinearization();
```

- Line 76 clear... Analysis at the begining of AW, constructed
- **Line 79 to 97, not clear! Study it. It seems to have an_depar and analysis file in FA to have the same output than MASTERODB. What is our output in OOPS then?**
- Line 100, nullification of pointers, clean up memory...

- **In Variational.h :**

```
83  //  Perform Incremental Variational Assimilation
84      IncrementalAssimilation<MODEL>(xx, *J, fullConfig);
85      Log::trace() << "Variational: incremantal assimilation done" << std::e
86
87  //  Save analysis and final diagnostics
88      if (!fullConfig.has("mimic_masterodb_oman")) {
89        PostProcessor<State_> post;
90        const util::DateTime winbgn(cfConf.getString("window_begin"));
91        const eckit::LocalConfiguration outConfig(fullConfig, "output");
92        post.enrollProcessor(new StateWriter<State_>(winbgn, outConfig));
93
94        const eckit::LocalConfiguration finalConfig(fullConfig, "final");
95        if (finalConfig.has("prints")) {
96          const eckit::LocalConfiguration prtConfig(finalConfig, "prints");
97          post.enrollProcessor(new StateInfo<State_>("final", prtConfig));
98        }
99
100       J->evaluate(xx, finalConfig, post);
101     }
102
103     return 0;
104   }
```

Preparation of the **Postprocessor** that will write the **analysis State, StateWriter.h.** Also the postprocessor **StateInfo** if demanded...

Line 100: **Total Cost Function J** is evaluated and **NL final forecast from beg AW through all AW is done and State written!** See CostFunction.h...

- **In CostFunction.h :**

```
239 double CostFunction<MODEL>::evaluate(const CtrlVar_ & fguess,
240                                       const eckit::Configuration & config,
241                                       PostProcessor<State_> post) const {
242 // Setup terms of cost function
243   PostProcessor<State_> pp(post);
244   JqTerm_ * jq = jb_->initialize(fguess);
245   pp.enrollProcessor(jq);
246   for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
247     pp.enrollProcessor(jterms_[jj].initialize(fguess));
248   }
249
250 // Run NL model
251   CtrlVar_ mfguess(fguess);
252   this->runNL(mfguess, pp);
253
```

- The list of **postprocessors** (+StateWriter) is increased by **initializing de different terms of J, Jb and Jo**, passing de analysis state. Jo initialize is though, we have gone through this before...
- We **runNL!!!** Along the AW. Go to CostFct3DVar.h...

- **In CostFct3DVar.h:**

```
128  void CostFct3DVar<MODEL>::runNL(CtrlVar_ & xx,
129                                  PostProcessor<State_> & post) const {
130    ASSERT(xx.state().checkStatesNumber(1));
131    ASSERT(xx.state()[0].validTime() == windowHalf_);
132    CostFct_::getModel().forecast(xx.state()[0], xx.modVar(), util::Duration(0), post);
133  }
```

- **This jumps to Model.h (we were here before):**

```
 96  void Model<MODEL>::forecast(State_ & xx, const ModelAux_ & maux,
 97                              const util::Duration & len,
 98                              PostProcessor<State_> & post) const {
 99    Log::trace() << "Model<MODEL>::forecast starting" << std::endl;
100    util::Timer timer(classname(), "forecast");
101
102    const util::DateTime end(xx.validTime() + len);
103    Log::info() << "Model:forecast: forecast starting: " << xx << std::endl;
104    this->initialize(xx);
105    post.initialize(xx, end, model_->timeResolution());
106    while (xx.validTime() < end) {
107      this->step(xx, maux);
108      post.process(xx);
109    }
110    post.finalize(xx);
111    this->finalize(xx);
112    Log::info() << "Model:forecast: forecast finished: " << xx << std::endl;
113    ASSERT(xx.validTime() == end);
114
115    Log::trace() << "Model<MODEL>::forecast done" << std::endl;
116  }
```

- We know this routine... And the F90 involved... For 3DVAR no 106-109 loop.
- In line 105 or line 111 should be called the postprocessor StateWriter... Where??

[redacted]

- **In CostFunction.h:**

```
254  // Cost function value
255    eckit::LocalConfiguration diagnostic;
256    if (config.has("diagnostics")) {
257      diagnostic = eckit::LocalConfiguration(config, "diagnostics");
258    }
259    double zzz = jb_->finalize(jq, config);
260    for (unsigned jj = 0; jj < jterms_.size(); ++jj) {
261      zzz += jterms_[jj].finalize(diagnostic);
262    }
263    Log::test() << "CostFunction: Nonlinear J = " << zzz << std::endl;
264    return zzz;
265  }
266
```

- In line 261 the scalar value of Jb and Jo is added to have the one of J. Nice to see how this calculations is done in method **finalize,** using **dot products...**

- **With this Variational.h terminated. Run.cc terminated and ifs4dvar.cc terminated! Good!!**

- **NOTE:**
  - **Not finished! We must write the analysis file… almost there…**
  - **It can be that some call to a F90 routine, particularly in a constructor list, is missing in this documentation**
  - **Use a underscore at the end of varname: var_ means that this variable is part of the private part of the class! As programmer convention…**

- **NOTE:**
  - **Outline the slides or steps/tasks that are particularly important for 4DVAR OOPS dev work…**
  - **Many intermediate calls (1.1.1.1…) can be ommited for comprehensiveness… In particular Factories adds 3 levels at least to the calling tree. Other may be repetitive (read file for setup geometry)**
  - **Refer code line by LXXX: L184. Try to set all important code lines with this reference in the slides**

[redacted]

- **NOTE:**
  - **Some jump counters elude “non important jumps”, like copying private members of objects”. E.g. 1.7.1.5.2.2.1.1.1.1 should be 1.7.1.5.2.2.1.1.1.2 but this does not introduce much info...**
  - **There are steps not understood yet from a theoretical basis point of view jump: E.g 1.7.1.5.2.2.1.1.1.3**
  - **Simplify the names of rutines where to jump! *File: routine***

[redacted]

- **NOTE:**
  - **All the ifs_xxxxx_c routines eventually call F90 IFS code repository rutines...**
  - **Not clear if stay at the end point in ifs_xxxx_c routines or also write down the pure F90 final ones (MODEL_CREATE.F90, ... Called from ifs_xxxx_c). Probably is better this!!!! For clarification, yes!**
  - **Needed small explanations on what is happening theoretically in the main tasks: e.g. Slide 55 on what does J.linearize**

- **FRAMEWORK CORE METHODS:**
  - **Ifs4dvar.cc (C++ main routine)**
  - **Variational.h: int execute()**
  - **IncrementalAssimilation.h: void IncrementalAssimilation()**
  - **SQRTMinimizer.h: doMinimize**
  - **SQRTPLanczosMinimizer.h: solve**

[redacted]

- **Array ordering in Fortran and C++**
  - In Fortran, array ordering is in *column major ordering*. In arguments, first columns, then rows. A declared array A(3,2) is:

A(1,1)	**A(2,1)**	**A(3,1)**
A(1,2)	**A(2,2)**	**A(3,2)**

  - In C++ array ordering is *row major ordering.* In arguments, first rows, then columns. Besides, in C++ array index starts in 0. A(3,2) is:

A(0,0)	**A(0,1)**
A(1,0)	**A(1,1)**
A(2,0)	**A(2,1)**

[redacted]

- **Pointers and arrays in C++**

```
// more pointers
#include <iostream>
using namespace std;

int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
  return 0;
}
```

- Arrays and pointers work very similar. In the expression left, when we do **“p = numbers”, p is storing the memory address of the first elements of array numbers *without using the & operator!.*** Then values for array numbers can be easily assigned through pointers...

- Here the ouput is 10, 20, 30, 40, 50

[redacted]

▪ Adding (and substracting) on pointers, not trivial!

that in a given system, `char` takes 1 byte, `short` takes 2 bytes, and `long` takes 4.

Suppose now that we define three pointers in this compiler:

```
1 char *mychar;
2 short *myshort;
3 long *mylong;
```

and that we know that they point to the memory locations `1000`, `2000`, and `3000`, respectively.

Therefore, if we write:

```
1 ++mychar;
2 ++myshort;
3 ++mylong;
```

`mychar`, as one would expect, would contain the value 1001. But not so obviously, `myshort` would contain the value 2002, and `mylong` would contain 3004, even though they have each been incremented only once. The reason is that, when adding one to a pointer, the pointer is made to point to the following element of the same type, and, therefore, the size in bytes of the type it points to is added to the pointer.

- **Increment (decrement) operator as sufix or prefix acts differently...** *Here operator "++" precedes operator "*"!!!*

```
1 *p++   // same as *(p++): increment pointer, and dereference unincremented address
2 *++p   // same as *(++p): increment pointer, and dereference incremented address
3 ++*p   // same as ++(*p): dereference pointer, and increment the value it points to
4 (*p)++ // dereference pointer, and post-increment the value it points to
```

- Basically, p++ (++ as sufix), address increment is applied **after** dereference *. And ++p (++ as prefix), address increment is applied **before** dereference... *It is more natural to me to think/use ++p...*

[redacted]

- **C++ allows to declare pointers to pointers!**

```
1 char a;
2 char * b;
3 char ** c;
4 a = 'z';
5 b = &a;
6 c = &b;
```

  - Here **b** is a pointer that stores the address where **a** is stored and **c** is a pointer that stores the address where pointer **b** is stored... **char ** c;** is the way to declare a pointer of a pointer...

- **Dereference operator ->**

```
struct movies_t {
  string title;
  int year;
};

int main ()
{
  string mystr;

  movies_t amovie;
  movies_t * pmovie;
  pmovie = &amovie;

  cout << "Enter title: ";
  getline (cin, pmovie->title);
```

- **-> operator** dereferences membre **title** of the pointer strucute **pmovie.** We can access to the memory pointed by pmovie.title...
- **pmovie -> == (*pmovie).** , because pmovie is a pointer.
- Be careful! *(pmovie.title) not possible since title is not a pointer here!!

- **Classes**

```
class class_name {
  access_specifier_1:
    member1;
  access_specifier_2:
    member2;
  ...
} object_names;
```

- **An access specifier is one of the following three keywords: private, public or protected.**
  These specifiers modify the access rights for the members that follow them:

- **private members** of a class are accessible only from within other members of the same class (or from their "friends").
- **protected members** are accessible from other members of the same class (or from their "friends"), but also from members of their derived classes.
- **public members** are accessible from anywhere where the object is visible.
- **By default members** (not declared differently) **are private!**

- **Scope operator ::**
  - It was used to access members of a **namespace**
  - It can be used in **classes** to access membres of the classe outside it. Useful to **define mtehods** for exemple...

```
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area () {return width*height;}
};

void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
```

- **Constructors**
  - A class **can** include a special function called its **constructor**, which is automatically called whenever a new object of this class is created, allowing the class to initialize member variables or allocate storage.
  - This constructor function is declared just like a regular member function, but **with a name that matches the class name** and without any return type; not even void.

[redacted]

- # Constructors

```
class Rectangle {
    int width, height;
  public:
    Rectangle (int,int);
    int area () {return (width*height);}
};

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

int main () {
  Rectangle rect (3,4);
  Rectangle rectb (5,6);
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
  return 0;
}
```

- **Rectangle(int,int)** is the constructor. It has the same name than the classe. It is executed always when the obect is declared (**rect**) and in this case sets the **private membres** of the class. This avoids **from the beggining** possible access to private data that is not initialized

Initialization of constructors

```
// classes and uniform initialization
#include <iostream>
using namespace std;

class Circle {
    double radius;
  public:
    Circle(double r) { radius = r; }
    double circum() {return 2*radius*3.14159265;}
};

int main () {
  Circle foo (10.0);   // functional form
  Circle bar = 20.0;   // assignment init.
  Circle baz {30.0};   // uniform init.
  Circle qux = {40.0}; // POD-like

  cout << "foo's circumference: " << foo.circum() << '\n';
  return 0;
}
```

- It is possible to initialize constructors using several sintaxis. 4 ways in the exemple on the left... All of them give the same result, so they are a **matter of style** (I don't like this diferent options to do the same thing...)

Constructor list

```
class Circle {
    double radius;
  public:
    Circle(double r) : radius(r) { }
    double area() {return radius*radius*3.14159265;}
};

class Cylinder {
    Circle base;
    double height;
  public:
    Cylinder(double r, double h) : base (r), height(h) {}
    double volume() {return base.area() * height;}
};

int main () {
  Cylinder foo (10,20);

  cout << "foo's volume: " << foo.volume() << '\n';
  return 0;
}
```

- Private members of the **main class** can also be constructed using a **constructor list.**

  Constructor list is a **list of constructors** of each private membre (they can be objects of other classes...), each with its arguments, that follows the name of the main class constructor, **separated by semicolon :**

Overload of operators

```
class CVector {
  public:
    int x,y;
    CVector () {};
    CVector (int a,int b) : x(a), y(b) {}
    CVector operator + (const CVector&);
};

CVector CVector::operator+ (const CVector& param) {
  CVector temp;
  temp.x = x + param.x;
  temp.y = y + param.y;
  return temp;
}

int main () {
  CVector foo (3,1);
  CVector bar (1,2);
  CVector result;
  result = foo + bar;
  cout << result.x << ',' << result.y << '\n';
  return 0;
}
```

- Operators like sum operator for exemple (+) can be **overloaded**, **changed its function** by defining a special method to do so. This method is a class method and is identified by using the keyword **operator**, before the operator whose meaning to be modified...
- **Here, "CVector operator + (const CVector&);" is not a constructor but a method of returning type CVector...**

The function `operator+` of class `CVector` overloads the addition operator (+) for that type. Once declared, this function can be called either implicitly using the operator, or explicitly using its functional name:

```
1 c = a + b;
2 c = a.operator+ (b);
```

[redacted]

Keyword this

```
class Dummy {
  public:
    bool isitme (Dummy& param);
};

bool Dummy::isitme (Dummy& param)
{
  if (&param == this) return true;
  else return false;
}

int main () {
  Dummy a;
  Dummy* b = &a;
  if ( b->isitme(a) )
    cout << "yes, &a is b\n";
  return 0;
}
```

- The keyword **this** represents **a pointer to the object whose member function is being executed**. It is used within a class's member function to refer to the object itself.
- Equivalent to **self** in Fortran2003 and Python

[redacted]

Static members of a class

```
class Dummy {
  public:
    static int n;
    Dummy () { n++; };
};

int Dummy::n=0;

int main () {
  Dummy a;
  Dummy b[5];
  cout << a.n << '\n';
  Dummy * c = new Dummy;
  cout << Dummy::n << '\n';
  delete c;
  return 0;
}
```

- A **static data member** of a class is a common variable **for all the objects** of that same class**, sharing the same value**: i.e., its value is not different from one object of this class to another
- They cannot be initialized directly in the class, but need to be initialized somewhere **outside** it:

  int Dummy::n=0;
- It exists the equivalent **static member functions..**