Geometric controller for COMPASS

Approach

The geometric controller is basically a projection of the input wavefront on the DM actuators :

$$c = PROJ.\Phi$$

with $PROJ = (IF^t.IF)^{-1}IF^t$ and IF the influence functions matrix of the DM.
This controller works in open-loop : this fact induces some modifications in the Yorick script to use it.

Algorithm

The algorithm is straight forward :

- Compute IF
- Filter the piston mode from Φ
- Compute the command

Compute IF

The only difficulty is due to the IF matrix : its size is Npts per Nactu where Npts is the number of points in the pupil. In ELT cases, this number can reach more than 500,000, causing an IF matrix and PROJ matrix of 10GB...

Nevertheless, this matrix is very sparse : by using the cuSPARSE Library, storing the matrix and performing the matrix-matrix multiplication is possible. It's not the case for the PROJ matrix, so instead of computing the complete PROJ matrix, only the matrix $\Gamma = (IF^t.IF)^{-1}$ is computed and stored. The inversion is done thanks to MAGMA Library.

Filter the piston

Φ is filtered of the piston mode by removing its average value across the pupil.

Compute the command

Since the PROJ cannot be computed, the command computation is performed in two steps :

$$A = IF^t.\Phi$$

and

$$c = -\Gamma.A$$

Yorick script

The geometric controller is open-loop based : the AO loop in the Yorick script has to be modified to take it into account. A basic AO loop for geometric controller is :

```
for (cc=1;cc<=y_loop.niter;cc++) {
        move_atmos,g_atmos;

        target_atmostrace,g_target,0,g_atmos;

        target_dmtrace,g_target,i-1,g_dm;

        sensors_trace,g_wfs,i-1,"atmos",g_atmos;

        sensors_trace,g_wfs,i-1,"dm",g_dm,0;

        sensors_compimg,g_wfs,i-1;

        rtc_docentroids,g_rtc,0;

        rtc_docontrol,g_rtc,0;

        rtc_applycontrol,g_rtc,0,g_dm;

}
```

Implementation details

New sutra_controller_geo class

```
class sutra_controller_geo: public sutra_controller {

public:

  long Nphi; // Number of points in the pupil

  carma_obj<double> *d_phi; // Input wavefront

  carma_obj<int> *d_indx_pup; // Index of where(pupil)

  carma_sparse_obj<double> *d_Ifsparse; // IF matrix sparse

  carma_obj<float> *d_geocov; // Γ matrix

  carma_obj<double> *d_compdouble; // Buffer in double for some computation

  carma_obj<float> *d_compfloat; // Buffer in float
public:
  sutra_controller_geo(carma_context *context, long nactu, long Nphi,
      long delay);
  sutra_controller_geo(const sutra_controller_geo& controller);
  ~sutra_controller_geo();
  string get_type();
  cusparseHandle_t cusparse_handle() {
                        return current_context->get_cusparseHandle();
            }
  int comp_dphi(sutra_source *target); //Retrieve Φ from the complete phase screen
  int comp_com(); // Compute the command
  int init_proj_sparse(sutra_dms *dms, int *indx_dm, float *unitpervolt, int *indx_pup); //
Compute matrix Γ
};
```

Results

See benchmark