Modal optimization for least square controller in COMPASS

Approach

This section summarizes the modal optimization approach for AO system command law described by E.Gendron & P.Léna, Modal control optimization, *Astronomical adaptive optics (1994).*

The phase of the instantaneous electric field over the pupil $\phi(x, y)$ can decomposed over some basis functions $M_i(x, y)$:

$$\phi(x,y) = \sum_{i=1}^{\infty} z_i.M_i(x,y)$$

In COMPASS, the modes M_i are considered as Karhunen-Loeve basis modes, but any other basis of modes may be of interest.
Then, if we note **z** the vector of z_i that described the input turbulent wavefront, we can write the wavefront **z'** corrected by the OA loop as :

$$\mathbf{z'} = \mathbf{z} - h_{ol}(f).G.D^{+}.D_{\infty}.\mathbf{z'} + h_{sys}(f).G.D^{+}.\mathbf{n}$$

with : h_{ol} and h_{sys} the transfer functions of the open-loop and the system respectively
G a diagonal filtering matrix that will be applied in addition to the control matrix
D is the interaction matrix
$\mathbf{n}$ is the noise vector on centroid position

Finally, by assuming that the aliasing term can be neglected in low flux regime, we can write each component of the vector **z** as :

$$z'_i = h_{cor}(f, g_i).z_i - h_n(f, g_i)\sum_j D^{+}_{ij}.n_j$$

And since the noise is not correlated with the signal, the variance is :

$$< z'^2 >= \int H_{cor}(f, g_i).\|z_i(f)\|^2 df + \int H_n(f, g_i).\|m_i(f)\|^2 df \qquad \text{(Equation 1)}$$

Where H_{cor} and Hn stand for the squared modulus of $h_{cor}(f)$ and $h_n(f)$ which are the correction transfer function and the transfer function between the white noise in input and the noise actually sent to the active component.
This equation shows that we can minimize the residual phase errors by a proper choice of the gain g_i and that is the goal of the modal control optimization.

Algorithm

Overall view

Here is the algorithm implemented in COMPASS and described step by step :

- Recording samples of open-loop measurements : in the RTC initialization, we runs a loop with turbulence, sensors imaging and centroiding in order to obtain samples of open-loop measurements. The number of recorded samples is set by the user in the parameter file with *y_controller.nrec*
- Computing the modes to volts matrix M2V : the pzt mirrors are decomposed on a Karhunen-Loeve basis
- Computing slopes to modes matrix S2M : $S2M = (D.M2V)^{-1}$

  It's important here to do the inversion without filtering modes. The matrix M2V has to be builded with a correct number of modes that allows to do this inversion. The user chooses the number of modes used with the parameter *y_controller.nmodes*
- Computing transfer functions : we compute $H_{cor}(f, g_i)$ for i=1 to i=*y_controller.ngain* and $y\_controller.gmin < g_i < y\_controller.gmax$
- Computing gain matrix G : computing Equation 1 and choosing g_i that minimizes the result
- Computing command matrix CMAT : $CMAT = M2V.G.S2M$
- Refreshing gains : during the loop, each closed-loop slope is reconstructed with a POLC equation and stored. When y_controller.nrec slopes are stored, we restart from step 5 to refresh modal gains in G.

M2V computation

The M2V matrix is computed by double diagonalization process.

First, the statistic covariance matrix C of the phase on the actuators is computed as :

$$C_{ij} = 6,88.\frac{r_{ij}}{r_0}^{5/3}$$

where r_{ij} is the distance between the actuator i and the actuator j

The piston mode is filtered from this matrix thanks to a filter matrix F (N = number of actuators) :

$$F = \begin{pmatrix} (1-\frac{1}{N}) & -\frac{1}{N} & \cdots & -\frac{1}{N} \\ -\frac{1}{N} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{1}{N} \\ -\frac{1}{N} & \cdots & -\frac{1}{N} & (1-\frac{1}{N}) \end{pmatrix}$$

and $C_{filtered} = F.C.F$

Then, the geometric covariance matrix Γ of the DM is computed as :

$$\Gamma = IF^t.IF$$

where IF is the influence functions matrix of the DM.
From those two matrix, the M2V matrix can be computed thanks to double diagonalization process :

- Diagonalize Γ : $M'^t.\Gamma.M' = D^2$
- Compute M : $M = M'.D^{-1}$
- Compute C' : $C' = M^{-1}.C.(M^t)^{-1}$
- Diagonalize C' : $A^t.C'.A = \Sigma$
- Compute M2V : $M2V = M.A$

Transfer functions computation

The transfer functions $H_{cor}(f, g_i)$ are computed analytically from the simulated system :

$$H_{cor}(f, g_i) = \| \frac{1}{1 + g_i.H_{ol}(f))} \|^2$$

with

$$H_{ol}(f) = \frac{1 - e^{-pTe}}{(p.Te)^2} e^{-p.t_{delay}} \text{ and } p = i2\pi f$$

This computation is done for each g_i and results are stored in a matrix H.

Modal gains computation

First, the decomposition of the recorded open-loop slopes **s** on the K-L basis is computed :

$$m = S2M.\mathbf{s}$$

Each vector ***m*** is then stored in a buffer (Nslope x Nmodes) and the PSD along each mode is computed:

$$PSD_i(f) = \|FFT(m_i)\|^2$$

Finally, Equation (1) is computed for each gain and the optimum gain for the mode *i* is the one that produces the minimum result.

$$G(i, i) = g_j / min(\sum PSD_i(f).H_{cor}(f, g_k)), k = 1, ..., ngain$$

POLC equation

After *y_controllers.nrec* iterations in the AO loop, the modal control optimization will be refresh from open-loop slopes reconstructed from the closed-loop ones thanks to a Pseudo Open-Loop Control equation.

If $s[k]$ stands for the estimation of the open-loop slope at iteration *k*, $y[k]$ for the closed-loop slope , $v[k]$ for the command applied to the DM and d the global delay in the loop, the estimation can be written as :

$$s[k] = y[k] - IMAT((d - 1).v[k - 2] + (2 - d).v[k - 1])$$

Those slopes are stored in a buffer Nslopes x nrec : when it's full, the modal control optimization is refreshed.

Implementation details

New parameters in Yorick controller structure

```
struct controller_struct
{
  [..]
  int     modopti;  // Flag for modal optimization
  int     nrec;     // Number of sample of open loop slopes for modal optimization computation
  int     nmodes;   // Number of modes for M2V matrix (modal optimization)
  float     gmin;     // Minimum gain for modal optimization
  float     gmax;     // Maximum gain for modal optimization
  int     ngain;    // Number of tested gains
};
```

New attributes in sutra_controller_ls

Those buffers are used during the AO loop, so they are declared as attributes of sutra_controller_ls but allocated only if modal ooptimization is used :

```
  carma_obj<float> *d_M2V; // Modes to Volts matrix (nactu x nmodes)

  carma_obj<float> *d_S2M; // Slopes to Modes matrix (nmodes x nslopes)

  carma_obj<float> *d_slpol; // Open-loop measurements buffer, recorded and loaded from Yorick
(nslopes x nrec)

  carma_obj<float> *d_Hcor; // Transfer function  (ngain x nrec/2)

  carma_obj<float> *d_com1; // Command k-1 for POLC (nactu)

  carma_obj<float> *d_com2; // Command k-2 for POLC (nactu)

  carma_obj<float> *d_compbuff; // Buffer for POLC computation (nactu)

  carma_obj<float> *d_compbuff2; // Buffer for POLC computation (nslopes)
```

The following attributes have been added to sutra_controller_ls also :

```
int is_modopti; // Flag for using modal optimization
int nrec; // Number of recorded open slopes measurements
int nmodes; //Number of modes
float gmin; // Gain min
float gmax; // Gain max
int ngain; // Number of tested gains between gmin and gmax
float Fs; // Sampling frequency
int cpt_rec; // Counter for modal gains refresh
```

New functions in sutra_controller_ls

```
int build_cmat_modopti(); //Compute the command matrix with modal optimization
int init_modalOpti(int nmodes, int nrec, float *M2V, float gmin, float gmax, int ngain,
                   float Fs); // Initialize arrays & compute S2M
int loadOpenLoopSlp(float *ol_slopes); // load open-loop slopes from Yorick
int modalControlOptimization(); // Compute optimum modal gains
int compute_Hcor(); // Compute transfer functions
```

New functions in sutra_dm

```
int get_IF_sparse(carma_sparse_obj<T> *&d_IFsparse, int *indx_pup, long nb_pts, float
ampli, int puponly); // Build the IF sparse matrix
template<class T>
int do_geomatFromSparse(T *d_geocov, carma_sparse_obj<T> *d_Ifsparse); // Compute
the geometric covariance matrix from IF sparse
int DDiago(carma_obj<float> *d_statcov, carma_obj<float>*d_geocov); // Double diago
int compute_KLbasis(float *xpos, float *ypos, int *indx, long dim, float norm, float ampli);
// Compute M2V matrix
int piston_filt(carma_obj <float> *d_statcov); // Filter the piston mode
```

Optimizations

- All matrix-vector or matrix-matrix multiplications are done using cuBLAS Library
- Inversion matrix is performed with MAGMA Library
- The influence functions matrix of the DM, which is required to compute the geometric covariance matrix usefull to obtain the M2V matrix, could be very large. In ELT cases for examples, it could be a (500,000 x 5,000) matrix, ie a 10 GB matrix. Knowing this matrix is very sparse, the cuSPARSE Libray is used to store the matrix in CSR format and to perform the matrix-matrix multiplication.

Results

See benchmark document

References

E.Gendron & P.Léna, *Modal control optimization*, *Astronomical adaptive optics (1994)*.

E.Gendron, *Optimisation de la commande modale en optique adaptative : application à l'astronomie*, *Thesis (1995)*