1 Supervectors and Notation

Let (A, B, C, D) be a l-input, m-output, state dimension n, linear, time-invariant, discrete, state space model of a dynamical system written in the form

$$x(t+1) = Ax(t) + Bu(t), x(0) = x_0,\ t = 0, \dots, N-1$$
$$y(t) = Cx(t) + Du(t),\ t = 0 \dots N.$$

Supervector description:

$$y = \begin{pmatrix} y(0) \\ y(1) \\ y(2) \\ \vdots \\ y(N) \end{pmatrix} \in \mathbb{R}^{m(N+1)}, u = \begin{pmatrix} u(0) \\ u(1) \\ u(2) \\ \vdots \\ u(N) \end{pmatrix} \in \mathbb{R}^{l(N+1)}.$$

$$y = Gu + d,$$

where

$$G = G(A, B, C, D) = \begin{pmatrix} D & 0 & \dots & 000 & & \\ CD & D & \dots & 000 & & \\ CAB & CB & \dots & 000 & & \\ \vdots & \vdots & & \vdots & \vdots & \\ CA^{N-2}B & CA^{N-2}B & \dots & CB & D & 0 \\ CA^{N-1}B & CA^{N-2}B & CA^{N-2}B & \dots & CB & D \end{pmatrix}$$

$$d = d(C, A, x_0) = \begin{pmatrix} Cx_0 \\ CAx_0 \\ CA^2x_0 \\ \vdots \\ CA^Nx_0 \end{pmatrix} \in \mathbb{R}^{m(N+1)}.$$

$r \in \mathbb{R}^{m(N+1)}$ denotes reference signal,

$$e_k = r - y_k,\ k \le 0$$

denotes the error on iteration k.

General formula for unit memory algorithm used here:

$$u_{k+1} = u_k + K_0 e_k \qquad \textbf{(The Input Update Rule)}, \tag{1}$$
$$\text{with } y_k = Gu_k + d \qquad \textbf{(Plant Dynamics)} \tag{2}$$

for some iteration matrix K_0 (the corresponding algorithm is named CGA, you can find it here).

2 Inverse Model Algorithms

The idea of the inverse algorithms is just to take the inverse of the system G and from the relation

$$y = Gu$$

and get the perfect u_∞ as

$$u_\infty = G^-1(r - y).$$

2.1 Right Inverse Model Algorithm

Input update low:

$$u_{k+1} = u_k + \beta G_R e_k, \tag{3}$$

where G_R denotes the right inverse of G and β iss a real scalar "learning gain".

Error evolution results in

$$e_{k+1} = (1 - \beta)e_k$$

or, equivalently,

$$e_{k+1} = (1 - \beta)^k e_0,$$

Algorithm obviously converges for $0 < \beta < 2$. For β close to 0 or 2, we get slow convergence with good robustness properties. For β close to unity we have rapidly convergence, but with reduced degree of robustness. For $\beta = 1$ we get convergence in one iteration. Implemented Algorithm you can find here.

2.2 Left Inverse Model Algorithm

Left inverse model algorithms results just by taking left instead of right inverse matrix.

Input update low:

$$u_{k+1} = u_k + \beta G_L e_k, \tag{4}$$

where G_L denotes the left inverse of G and β is a real scalar "learning gain".

Error evolution results in

$$e_{k+1} = (1 - \beta)e_k$$

or, equivalently,

$$e_{k+1} = (1 - \beta)^k e_0,$$

Implemented Algorithm you can find here.

It is worth to say that the left inverse model algorithm doesn't always return a perfect solution: if the initial error doesn't lie in the image of G, the error converges to e_∞, which is the projection of e_0 onto the kernel of $(I - GG_L)$:

$$\lim_{k\to\infty} e_k = e_\infty = (I - GGL)e0.$$

2.3 Robustness

Wird ergänzt. Die Algorithmen habe ich eher noch nicht getestet.

We consider the left multiplicative uncertainty model $H := UG$. Then the uncertain error evolution results in

$$e_{k+1} = (I - \beta U)e_k. \tag{5}$$

Theorem 1. *Consider the Right Inverse Model algorithm with β in range of $0 < \beta < 2$. Then, a necessary and sufficient condition for robust monotonic convergence in the presence of the left multiplicative modeling error U is that*

$$\|(I - \beta U)\| \leq 1.$$

There are also other tests, which I'll write down later. The main idea is that if $\|I - \gamma U\| \leq \delta$ we can choose β in some range $\beta \in (0, \beta^*)$ for some $\beta^* > 0$, and guarantee robust convergence. The lower bound for β^* is $\beta^{**} = \gamma\left(\frac{1-\delta}{1+\delta}\right)$.

2.4 Relaxation

Kommt noch, lässt Robustheit verbessern

2.5 Robustness and Non-monotonic Convergence

Kommt noch, es geht um die Gewichtung des Systems mit einem epsilon, sodass die Voraussetzungen an die robuste Konvergenz erfüllt sind.

3 Gradient Algorithms

Gradient algorithms are another example of methods, which achieve monotonic convergence.

3.1 The Steepest Descent Algorithm for Iterative Control

Input update rule:

$$u_{k+1} = u_k + \beta G^* e_k, \tag{6}$$

error evolution:

$$e_{k+1} = Le_k = L^k e_0,\ L = I - \beta GG^*. \tag{7}$$

G^* denotes here conjugate transpose of the matrix G, β is as before a "learning gain".

β must be chosen between 0 and $\frac{2}{\|G^*\|^2}$ to insure the convergence. In particular, for $0 < \beta < \frac{2}{\|G^*\|^2}$ and for any initial error e_0:

1. $$\lim_{k\to\infty} e_k = e_\infty = P_{ker[G^*]e_0}, \tag{8}$$

   where $P_{ker[G^*]e_0}$ is the self adjoint, positive definite, orthogonal projection operator onto $ker[G^*]$.

2. If the initial error $e_0 \in \overline{\text{Image}[GG^*]}$, then

$$\lim_{k\to\infty} = 0. \tag{9}$$

Implemented algorithm you can find here.

3.2 Suppression of Eigenvalues

In steepest descent algorithms we've chosen gain β as constant. It allows much more flexibility, if we set for each iteration different β_k. Especially if we let β_k depend on eigenvalues of GG^*. Next theorem give an insight, why it could be a good choice.

Theorem 2. *Let $\lambda_{j1}, \lambda_{j2}, \lambda_{j3}, \ldots$ be a chosen ordering of the non-zero eigenvalues of GG^* and suppose that GG^* has q zero eigenvalues. Then the parameter varying iterative control algorithm*

$$u_{k+1} = u_k + \beta_{k+1} G^* e_k, \quad \textit{with the choice } 1 - \beta_{k+1}\lambda_{j_{k+1}} = 0,$$

converges to the limit e_∞ defined by (1) in a finite number of iterations.

The problem with this algorithm is that small values of λj will lead to very large values of β_j. To stay within the range of convergence, we only can use the gains in the range of $0 < \beta_k \|G\|^2 < 2$, which is satisfied, only if $\lambda_j < \frac{2}{\|G\|^2}$. In this case the algorithm can, in principle, be used to eliminate all components with eigenvalues satisfying this inequality.

Though this algorithm is illustrated only as theoretical motivation, I've implemented it to see how significant is the difference between the following algorithm and this one. The implementation you can find here.

Algorithm 1. *Choose a finite number N_p of points $p_1, p_2, \ldots$ spread over the half-open interval $(\|G\|^2, \|G\|^2]$ and set the gain $\beta_k = p_k^{-1}$ on any iteration $k \le N_p$. If, on subsequent iterations, a fixed value in the range of $0 < \beta\|G\|^2 < 2$ os used, it follows, that the resultant error sequence has the form*

$$e_k = \left(\prod_{j=1}^{k} (I - \beta_j GG^*)\right) e_0, \quad \textit{for } 1 \le k \le N_p, \ \textit{and} \tag{10}$$

$$e_k = (I - \beta GG^*)^{(k-N_p)} e_{N_p}, \quad \textit{for } k \ge N_p. \tag{11}$$

We can consider e_{N_p} here as the initial error for the remainder of the iterative process. For large N_p, the approximation is good enough to return much better results as classical steepest descent algorithm. The implementation you can find here.

4 Model Reducing

Hier habe ich ein Problem :(Kap.7.5.1

In the steepest descent algorithm we've chosen $K_0 = G^*$. This time let's take $K_0 = K^*$ for some matrix K, which is simpler than, but not necessarily the same as G. "Typically, it could be envisaged that, by choosing K as simplified model of G with reduced state dimension, the computational load when computing the input signal u_{k+1} from u_k and e_k will be reduced whilst simultaneously satisfying other conditions (such as those discussed in what follows) that ensure monotonic convergence " (c)Book

Algorithm 2. *Using the notation defined above, an Iterative Algorithms defined above using the adjoint K^* of a system (A_K, B_K, C_K, D_K), has the* ***input update rule:***

$$u_{k+1} = u_k + \beta K^* e_k. \tag{12}$$

It is ***assumed*** *that both G and K are asymptotically stable and that $rank(D) = rank(D_K) = \min\{m, l\}$.*

In case $m \leq l$ if

$$GK^* + KG^* > \beta KG^*GK^* \text{ (eq. 7.70 in the book)} \tag{13}$$

the monotonic convergence is guaranteed.

In case $m \geq l$ monotonic convergence to 0 is guaranteed if

$$K^*G + G^*K > \beta G^*KK^*G. \text{ (eq. 7.95 in the book)} \tag{14}$$

Im Buch betrachtet er $G = KU$ für irgeneine Matrix U., schreibt es aber nicht in die Voraussetzungen rein (s. 185). Kann man das eigentlich für balred garantieren? Ich schaue mir das Kapitel nochmals an, wahrscheinlich habe ich da igendwas verapasst.

4.1 Robustness

4.2 Relaxaion

Kommt noch

4.3 ε-Weighted Norms

Kommt noch