UKACC International Conference on Control 2012
Cardiff, UK, 3-5 September 2012

Frequency-Domain Tuning of Fixed-Structure Control Systems

Pascal Gahinet and Pierre Apkarian

Abstract— **This paper presents two new MATLAB-based tools for tuning fixed-structure linear control systems in the frequency domain: `hinfstruct` and `looptune`. These tools can directly tune control architectures with multiple feedback loops and multiple fixed-order, fixed-structure control elements. The controller parameters are tuned using non-smooth H_∞ optimization but little a-priori knowledge of the H_∞ theory is required. This makes such tools ideally suited for real-world applications where the control system structure and complexity are constrained. An application to helicopter control is discussed and the results of an extensive benchmark of `hinfstruct` vs. `hifoo` are reported.**

I. Introduction

H_∞ theory [1], [2], [3], [4] provides powerful techniques for synthesizing controllers in the frequency-domain. Typical design requirements such as speed of response, control bandwidth, disturbance rejection, and robust stability are naturally expressed as constraints on the gain (H_∞ norm) of well-chosen closed-loop transfer functions. In turn, efficient algorithms and software tools are available to synthesize MIMO controllers that satisfy such gain constraints [1], [5], [6].

Yet existing H_∞ synthesis tools have practical limitations that have slowed their adoption in industry. H_∞ controllers are monolithic whereas most embedded control architectures are decentralized collections of simple control elements such as gains and PID controllers. H_∞ controllers tend to be opaque and complex (high number of states) whereas embedded controllers tend to be intuitive and have low complexity. And recasting the design requirements as a single aggregate H_∞ constraint can be challenging for engineers. As a result, hand tuning and optimization-based tuning tend to remain the norm for decentralized control systems.

This paper presents new MATLAB tools for *Structured H_∞ Synthesis* [6] that overcome the limitations listed above. These tools leverage state-of-the-art nonsmooth optimizers [7], [8] to directly and efficiently tune arbitrary control architectures. By "arbitrary," we mean any single- or multiple-loop block diagram arrangement containing any number and type of linear control elements, from simple gains and PIDs to more complex notch filters and state-space controllers. Some of these tools also automate the H_∞ formulation, allowing users to tune the controller elements directly from high-level specifications. Finally, despite the lack of convexity, these tools perform well in practice, both in terms of speed of execution and quality of the solutions.

P. Gahinet is with MathWorks, 3 Apple Hill, Natick, MA 01760-2098, USA `Pascal.Gahinet@mathworks.com`

Pierre Apkarian is with ONERA and Institut de Mathématiques, Université Paul Sabatier, 2, av. Ed. Belin, 31055, Toulouse, France `Pierre.Apkarian@onera.fr`

The paper is organized as follows. Section 2 discusses the standard formulation of structured H_∞ synthesis and the representation of tunable control elements. Section 3 presents `hinfstruct`, a general-purpose tool for structured H_∞ synthesis. Section 4 presents `looptune`, a more specialized tool that automates mainstream tuning tasks. Finally, Section 5 gives an application example and Section 6 reports the results of a comparison with `hifoo`.

II. Framework for Tuning Fixed-Structure Control Systems

As mentioned earlier, our starting point is any linear control architecture with one or more fixed-structure blocks to tune, for example the feedback structure shown in Figure 4 where the shaded blocks are tunable. Since there are infinitely many possible architectures, we use a formal representation called *Standard Form* that is both general and convenient to work with. The Standard Form is depicted in Figure 1 and consists of two main components:

- An LTI model $P(s)$ that combines all fixed (non tunable) blocks in the control system
- A structured controller $C(s) = \mathrm{Diag}(C_1(s), \ldots, C_N(s))$ that combines all tunable control elements. Each control element $C_j(s)$ is assumed to be linear time invariant and to have some prescribed structure.

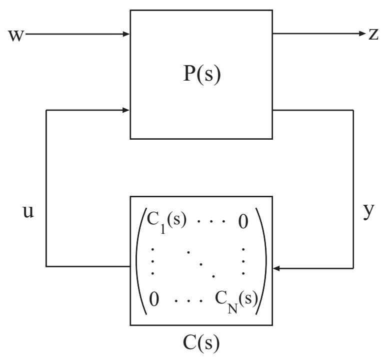

w
z
P(s)
u
y
$\begin{pmatrix} C_1(s) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & C_N(s) \end{pmatrix}$
C(s)

Fig. 1. Standard Form for Structured H_∞ Synthesis

External inputs to the system such as reference signals and disturbances are gathered in w and performance-related outputs such as error signals are gathered in z. The closed-loop transfer function from w to z is given by the linear-fractional transformation (LFT) [9]:

$$T_{zw}(s) = F_l(P, C) := P_{zw} + P_{zu}C(I - P_{yu}C)^{-1}P_{yw}. \quad (1)$$

It is well known from Robust Control theory that any block diagram can be rearranged into this Standard Form by isolating the tunable blocks and collapsing the rest of the diagram into $P(s)$. The resulting model has the same structure as the uncertain models used in μ analysis (with the uncertainty blocks $\Delta_j(s)$ replaced by the control elements $C_j(s)$) [10], [11], [12]. Figure 1 is also reminiscent of standard H_∞ synthesis [1] but differs in one key aspect, namely, the special structure of the controller $C(s)$. Since systematic procedures and automated tools are available to transform any architecture to the Standard Form of Figure 1, we henceforth assume that the control architecture is specified in this form.

The next challenge is to describe the tunable control elements. Again we are faced with a wide range of possible structures, from simple gains and PIDs to more complex lead-lag and observer-based controllers. Since our approach is based on optimization, it is natural to use parameterizations of such components. For example, a PID can be parameterized by four scalars K_p, K_i, K_d, T_f as

$$C_j(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s + 1}. \qquad (2)$$

Similarly, a state-space controller with fixed order n can be parameterized by four matrices A, B, C, D of suitable sizes. To parameterize more general elements like the lowpass and notch filters

$$\frac{a}{s+a}, \quad \frac{s^2 + 2\zeta_1\omega_0 s + \omega_0^2}{s^2 + 2\zeta_2\omega_0 s + \omega_0^2}, \qquad (3)$$

we introduce a basic building block called "real parameter" (`realp`). If a is a real parameter, then any well-posed rational function $R(a)$ can be written as the LFT:

$$R(a) = F_l(M, a \otimes I) \qquad (4)$$

where M is a fixed matrix and $a \otimes I := \mathrm{Diag}(a, \ldots, a)$ [13]. Since any interconnection of LFT models is an LFT model, it is easily seen that if the control element $C_j(s)$ is a rational function of a, the Standard Form of Figure 1 can be rearranged so that $C_j(s)$ is replaced by a block-diagonal matrix with copies of a on the diagonal. This remains true when $C_j(s)$ depends on multiple parameters $a_1, \ldots, a_M$. In other words, we can absorb the specific structure of $C_j(s)$ into $P(s)$ and keep only the low-level tunable parameters $a_1, \ldots, a_M$ in the $C(s)$ block of Figure 1. Note that unlike μ-analysis, "repeated" blocks do not affect the optimization outcome and only incur some small overhead.

For example, consider the lowpass filter $F(s) = \frac{a}{s+a}$ where a is tunable. This tunable element is specified by:

```
a = realp('a',1);  % a initialized to 1
F = tf(a,[1 a]);   % creates a/(s+a)
```

This automatically builds the following LFT representation of $F(s)$:

$$F(s) = F_l\left(\left(\begin{array}{ccc} 0 & 0 & 1 \\ 1/s & -1/s & 0 \\ 1/s & -1/s & 0 \end{array}\right), \left(\begin{array}{cc} a & 0 \\ 0 & a \end{array}\right)\right). \qquad (5)$$

Note that while it is possible to parameterize $F(s)$ using a single copy of `a`, the convenience of the syntax shown above more than makes up for the small overhead incurred by the extra copies of a.

III. Structured H_∞ Synthesis

Now that we have a framework for describing arbitrary control architectures and linear control elements, we turn to the question of using H_∞ synthesis to tune the controller parameters in the Standard Form of Figure 1. H_∞ synthesis is a frequency-domain method for enforcing typical control design requirements. At the heart of the method is the H_∞ norm, which measured the peak input/output gain of a given transfer function:

$$\|H(s)\|_\infty := \max_\omega \overline{\sigma}(H(j\omega)). \qquad (6)$$

In the SISO case, this norm is just the peak gain over frequency. In the MIMO case, it measures the peak 2-norm of the frequency response $H(j\omega)$ over frequency.

It is well-known from robust control theory [14] that classical design requirements (bandwidth, roll-off, disturbance attenuation, stability margins) can be recast as normalized H_∞ constraints of the form

$$\|W_j(s)T_j(s)\|_\infty < 1, \quad j = 1, \ldots, M \qquad (7)$$

where the T_j's are suitable closed-loop transfer functions and the W_j's are weighting functions that reflect the nature and parameters of each requirement. So a typical controller tuning task consists of adjusting the controller parameters to satisfy the constraints (7). Introducing

$$H(s) := \mathrm{Diag}(W_1(s)T_1(s), \ldots, W_M(s)T_M(s)). \qquad (8)$$

(7) is equivalent to $\|H(s)\|_\infty < 1$. Since each transfer function T_j can be expressed as an LFT model depending on the structured controller $C(s) := \mathrm{Diag}(C_1(s), \ldots, C_N(s))$, the Standard Form of $H(s)$ looks like

$$H(s) = F_l\left(P(s), \mathrm{Diag}(C(s), \ldots, C(s))\right) \qquad (9)$$

where $P(s)$ is a fixed LTI model. So *independently* constraining two or more closed-loop transfer functions $T_j(s)$ as in (7) leads to repeating the controller $C(s)$ multiple times in the Standard Form. The resulting block-diagonal controller structure is beyond the scope of standard H_∞ algorithms but poses no problem in our framework since this merely amounts to repeating the tunable blocks along the diagonal (see Section II for a discussion of repeated blocks). This is an important advantage over traditional H_∞ synthesis where all requirements must be expressed in terms of a *single* MIMO closed-loop transfer function $T(s)$.

Summing up, decentralized controller tuning can be recast as a structured H_∞ synthesis problem where the controller has a block-diagonal structure, each block being parameterized and possibly repeated. In turn, we can use the nonsmooth algorithms described in [7], [15], [16] to optimize the controller parameters and enforce the constraint $\|H(s)\|_\infty < 1$. The MATLAB sofware described here is based on [7], [17] and consists of three main components:

- Simple objects to specify tunable parameters (realp) and elementary control elements such as gains, low-order transfer functions, and PIDs
- Overloaded arithmetic and interconnection algebra to automatically build the Standard Form of $H(s)$ by combining/connecting together ordinary LTI models, tunable elements, and weighting functions
- The hinfstruct function for minimizing the H_∞ norm of $H(s)$ with respect to the tunable controller parameters. This function can be seen as the counterpart of hinfsyn for structured H_∞ synthesis

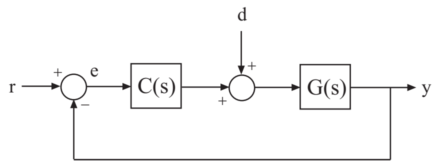


Fig. 2. Elementary feedback loop.

To get a feel for these tools, consider the simple scenario where the requirements for the feedback loop of Figure 2 can be expressed as

$$\|w_S S\|_\infty < 1, \quad \|w_T T\|_\infty < 1 \tag{10}$$

where $S = 1/(1+L)$, $T = L/(1+L)$, $L = GC$, and w_S, w_T are suitable frequency-weighting functions. Also assume that $C(s)$ is constrained to be a PID controller. Using the sofware, you can construct a parametric model of $H(s) = \text{Diag}(w_S S, w_T T)$ as follows:

```
G = tf([1 2],[1 5 10]);      % plant model
C = ltiblock.pid('C','pid'); % tunable PID
S = feedback(1,G*C);
T = feedback(G*C,1);
H0 = blkdiag(wS * S, wT * T);
```

The result H0 is a MATLAB representation of the (untuned) Standard Form for $H(s)$ and depends on the tunable PID block C. Next invoke hinfstruct to tune the PID controller gains so as to enforce $\|H(s)\|_\infty < 1$:

```
H = hinfstruct(H0);
```

The output H contains the tuned Standard Form of $H(s)$ and you can access the tuned value of the PID controller C with

```
C = getBlockValue(H,'C')
```

Note that hinfstruct actually minimizes the H_∞ norm of $H(s)$ but can be configured to terminate as soon as the target value of 1 is achieved. Also, hinfstruct can be configured to automatically run multiple optimizations from randomly generated starting points. This helps mitigate the local nature of the optimizer and increases the likelihood of finding parameter values that meet the design requirements. See [18], [6], [19] for more details and examples.

IV. Automated Tuning of Feedback Loops

While hinfstruct addresses the first two practical limitations of traditional H_∞ synthesis tools, familiarity with the H_∞ methodology is still required to turn typical design specifications into a well-posed H_∞ optimization problem.

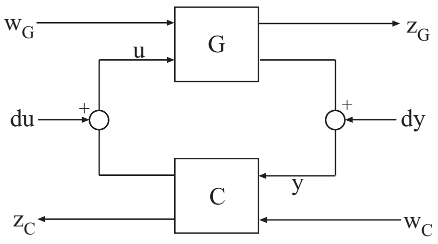


Fig. 3. Generic MIMO feedback loop.

This difficulty is exacerbated in multi-loop control systems because of scaling and coupling issues. For example, a poor choice of units in one feedback channel may skew the sensitivity function and lead to an ill-posed H_∞ problem [14, p. 5-8]. Also, classical one-loop-at-a-time stability margins may be misleading when cross-coupling exists between feedback loops [20]. Such challenges led us to seek ways to automate the H_∞ formulation of high-level requirements. This turns out to be possible and to often lead to satisfactory results. We now discuss this "push-button" approach and the looptune function that embodies it.

Our starting point is the generic MIMO feedback loop of Figure 3 where G represents the "plant" and C represents the overall controller. Any control structure can be rearranged in this fashion by using the measurement signals y and control signals u to separate the controller from the plant. Both G and C may contain tunable elements, which allows for co-tuning of plant and controller parameters. To formulate an H_∞ synthesis problem for this feedback system, observe that most controller tuning tasks involve some combination of the following requirements:

1) **Performance**: The feedback loops should have high gain at low frequency to reject disturbances and follow setpoint changes
2) **Roll off**: The feedback loops should have low gain at high frequency to guard against unmodeled dynamics and measurement noise
3) **Stability**: The feedback loops should be stable with enough margin to sustain typical amounts of gain and phase variations at the plant inputs and outputs.

Typically, the first two requirements amount to shaping the open-loop response to have integral action at low frequency and roll off in excess of -20 dB/decade at high frequency. The transition from high to low open-loop gain occurs in the *gain crossover band* (an interval in the MIMO case since in general it is neither possible nor desirable to make all loops cross at the same frequency). The gain crossover band determines the response time and bandwidth of the control system. Using the standard "mixed-sensitivity" formulation, we can express these loop-shaping requirements as

$$\left\| \begin{pmatrix} W_{LF} S_i \\ W_{HF}(I - S_i) \end{pmatrix} \right\|_\infty < 1 \tag{11}$$

where S_i is the sensitivity function at the plant inputs u and the weighting functions W_{LF}, W_{HF} reflect the desired loop shape. Note that S_i should be replaced by the sensitivity

S_o at the plant outputs if there are more controls u than measurements y.

For the stability requirement, we use the notion of multivariable disk margins discussed in [20]. This measure guarantees robustness against simultaneous gain and phase variations at all plant inputs and outputs, which is much stronger than one-loop-at-a-time stability margins. With the notation

$$L(s) = \begin{pmatrix} 0 & G(s) \\ C(s) & 0 \end{pmatrix}, \quad X(s) = (I + L)(I - L)^{-1}, \tag{12}$$

the robust stability condition is

$$\mu(X(s)) < 1/\alpha \tag{13}$$

where $\mu(.)$ denotes the structured singular value for a diagonal block structure [4] and the parameter α is a function of the desired gain and phase margins [20]. For tractability reasons, we replace this condition by:

$$\min_{D} \max_{\omega \in [\omega_1, \omega_2]} \|D^{-1} X(jw) D\| < 1/\alpha \tag{14}$$

where D is a constant and diagonal scaling matrix and $[\omega_1, \omega_2]$ is some interval containing the gain crossover band. The rationale for this simplification is that (a) stability margins are worst near the gain crossovers, and (b) the gain crossover band is typically narrow enough that we can get away with a constant rather than frequency-dependent D-scaling in the $\mu(.)$ upper bound.

Note that the scaling $D = \begin{pmatrix} D_o & 0 \\ 0 & D_i \end{pmatrix}$ is equivalent to the plant I/O scaling $G \to D_o^{-1} G D_i$. In other words, D automatically corrects scaling issues in the vector signals u and y, e.g., u having both small and large entries due to a poor choice of units. Because the H_∞ norm is not invariant under I/O scaling, this turns out to be essential to formulate a meaningful H_∞ synthesis problem [14, Remark 1]. Finally, (14) is tractable in our framework if we treat the diagonal entries of D as tunable parameters ($D^{-1}X(jw)D$ is an LFT in the controller and scaling matrix D).

Summing up, for a given crossover frequency/band, tuning the controller amounts to finding a scaling D and controller parameter values that satisfy

$$\left\| \begin{pmatrix} W_{LF} D_i^{-1} S_i D_i \\ W_{HF}(I - D_i^{-1} S_i D_i) \end{pmatrix} \right\|_\infty < 1 \tag{15}$$

$$\max_{\omega \in [\omega_1, \omega_2]} \|\alpha D^{-1} X(jw) D\| < 1. \tag{16}$$

Note that we use the scaled input sensitivity $D_i^{-1} S_i D_i$ instead of S_i to take advantage of the u scaling provided by D.

The `looptune` function [6] formulates and solves this H_∞ optimization problem. The basic interface is

```
[G,C] = looptune(G0,C0,wc,Req1,Req2,...)
```

where `G0,C0` are (untuned) parametric models of G and C, `wc` is the target crossover frequency/band, and the optional arguments `Req1,Req2,...` specify additional requirements such as maximum gain or setpoint tracking. Note that `wc` can be omitted and replaced by more sophisticated loop shaping requirements, thus providing a fair amount of flexibility.

V. Helicopter Example

This section presents an application to a challenging helicopter control problem. We use an 8-state model of the Westland Lynx helicopter at the hovering trim condition. The controller generates commands d_s, d_c, d_T in degrees for the longitudinal cyclic, lateral cyclic, and tail rotor collective using measurements of θ, ϕ, p, q, r (pitch and roll angles and roll/pitch/yaw rates). For details and data, see [21] and the demo in [6]. The controller structure is shown in Figure 4 and consists of two feedback loops:

- The inner loop (static output feedback SOF) provides stability augmentation and decoupling
- The outer loop (PI controllers PI1-PI3) provides the desired setpoint tracking performance.

The main control objective is to track setpoint changes in θ, ϕ, r with zero steady-state error, settling times of about 2 seconds, minimal overshoot, and minimal cross-coupling.

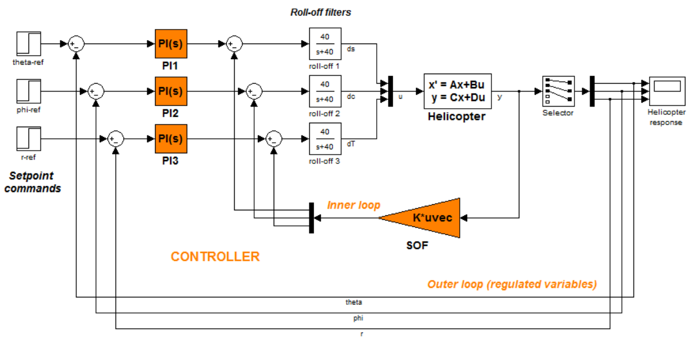


Fig. 4. Control architecture for Westland Lynx helicopter.

Since the helicopter is modeled in Simulink we use the `slTunable` interface to quickly set up the `looptune` optimization. With this interface you just specify the Simulink blocks to tune, the measurement and control signals (controller I/Os), and the I/O signals of interest for closed-loop analysis:

```
ST0 = slTunable('helico',...
                {'PI1','PI2','PI3','SOF'});
ST0.addControl('u')
ST0.addMeasurement('y')
ST0.addIO({'theta_ref','phi_ref','r_ref'},'in')
ST0.addIO({'theta','phi','r'},'out')
```

This information is used to automatically parameterize the tuned blocks and linearize the Simulink model to extract the plant model G and a parametric model of the controller C. Note that the static-output-feedback gain is initialized to zero and the PI controllers to $1 + 1/s$, values for which the closed-loop response is unstable.

We want the outer loop to settle in about 2 seconds so the open-loop bandwith should be at least 2 rad/s (based on first-order characteristics). The inner loop must typically be faster so we seek a gain crossover band between 10 and 30 rad/s:

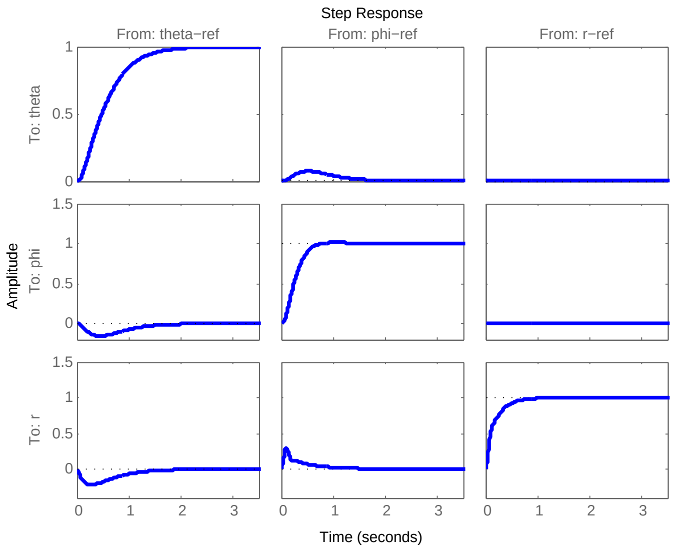


Fig. 5. Closed-loop responses to θ, ϕ, r commands.

```
wc = [10,30];
```

Because there are fewer actuators (3) than measurements (5), integral action in the open-loop response is not enough to guarantee that θ, ϕ, r will track the setpoint commands $\theta_{\rm ref}, \phi_{\rm ref}, r_{\rm ref}$. We therefore add an explicit tracking requirement with a 2-second response time:

```
TR = TuningGoal.Tracking(...
        {'theta_ref','phi_ref','r_ref'},...
        {'theta','phi','r'},2);
```

Finally, we specify the desired minimum gain and phase margins:

```
Opt = looptuneOptions('GainMargin',5,...
                               'PhaseMargin',40);
```

We can now tune the controller parameters with `looptune`:

```
ST = ST0.looptune(wc,TR,Opt);
```

The final H_∞ norm is 1.28, again close to 1, and the closed-loop step responses are shown in Figure 5. These responses settle in less than two seconds with no overshoot and small cross-coupling. The tuned values are:

$$\begin{aligned} PI_1(s) &= 0.45 + 12.2/s \\ PI_2(s) &= -0.17 - 9.15/s \\ PI_3(s) &= -0.66 - 8.22/s \\ SOF &= \begin{pmatrix} 7.24 & -0.84 & -0.0031 & 1.02 & -0.045 \\ -1.30 & -2.45 & 0.01 & -0.14 & -0.21 \\ -1.43 & 0.81 & -1.47 & -0.23 & 0.17 \end{pmatrix} \end{aligned}$$

Tuning these 21 parameters starting from an unstable initial guess took 11 seconds on a 64-bit PC with a 3 GHz dual-core processor and 6 GB of RAM.

VI. BENCHMARK

This section reports the result of a comprehensive benchmark of `hinfstruct` vs. `hifoo`. Both `hifoo` and `hinfstruct` implement state-of-the-art nonsmooth programming techniques. `hifoo` is a two-stage technique where a smooth linesearch BFGS approach is followed by nonsmooth gradient sampling. This means gradients are randomized around the current iterate to refine or establish optimality in the second phase. `hinfstruct` exploits extension sets of the Clarke sub-differential at each iteration and derives a tangent subproblem in the form of a nonsmooth convex QP approximation of the original problem. A search direction is then computed and a linesearch is carried out. `hinfstruct` is fully deterministic and does not use randomization except for (optionally) choosing the starting point. Both techniques have local optimality certificates.

Our assessment is based on 234 test cases from the *COMP*l$_{\rm e}$*ib* benchmark library [22] and compares the following versions:

- `hinfstruct` from [6]
- `hifoo 3.5` with `hanso 2.1` [15].

Details on the test cases can be found at [19]. Both codes are run in default mode with 3 starting points in each case. For `hifoo` we run the gradient sampling phase to enhance accuracy and provide an optimality certificate.

A graphical comparison of the achieved objective values and execution times for both techniques appears in Figure 6. Recall that the objective value is the closed-loop H_∞ norm. The top plot shows a bar chart of the H_∞-norm log-ratios:

$$\log_2(H_\infty\text{-norm } \texttt{hinfstruct}/H_\infty\text{-norm } \texttt{hifoo})\,.$$

Bars to the left of the vertical line $x = 0$ indicate that `hinfstruct` terminated with a lower H_∞-norm than `hifoo`, and bars to the right indicate the opposite. A bar of unit length materializes improvement by a factor 2, a bar of length 2 improvement by a factor of 4, etc. Similarly, the bottom plot shows a bar chart of the CPU time log-ratios:

$$\log_{10}(\text{cpu time } \texttt{hinfstruct}/\text{cpu time } \texttt{hifoo})\,.$$

Here a bar of unit length means 10 times faster, a bar of length 2 means 100 times faster, etc.

Further comparison of CPU times for problems where `hinfstruct` and `hifoo` terminate with essentially the same objective value is shown in Figure 7. These results show that `hinfstruct` is significantly faster and often more accurate than `hifoo`.

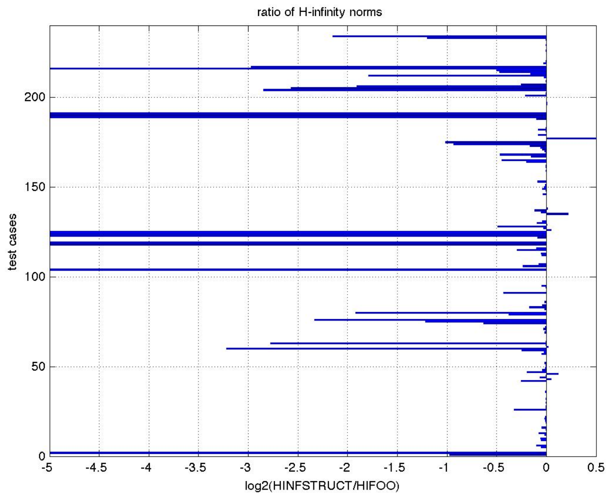
CONCLUSION

We have presented a new methodology and tool set for tuning fixed-structure SISO or MIMO control systems. While our approach is rooted in H_∞ theory, it is clear that these tools and techniques are not restricted to Robust Control applications and can be seen as a systematic framework for tuning decentralized control architectures and exploring the trade-offs between performance and complexity.

REFERENCES

[1] J. Doyle, K. Glover, P. P. Khargonekar, and B. A. Francis, "State-space solutions to standard H_2 and H_∞ control problems," *IEEE Trans. Aut. Control*, vol. AC-34, no. 8, pp. 831–847, Aug. 1989.

[2] G. Stein and J. C. Doyle, "Beyond singular values and loop shapes," *J. Guidance and Control*, vol. 14, pp. 5–16, 1991.



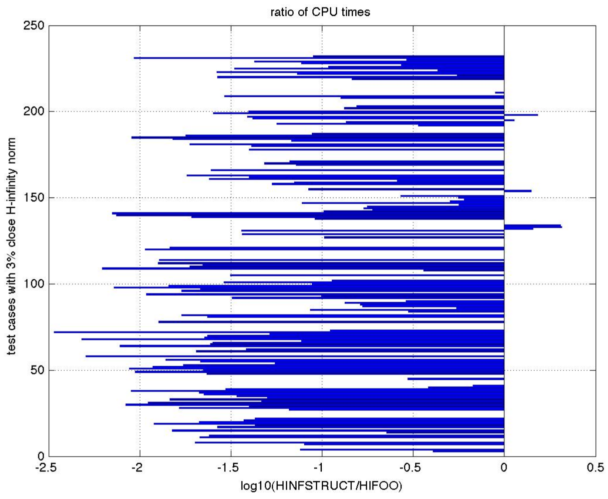


Fig. 7. Comparison of CPU times when final values differ by 3% or less

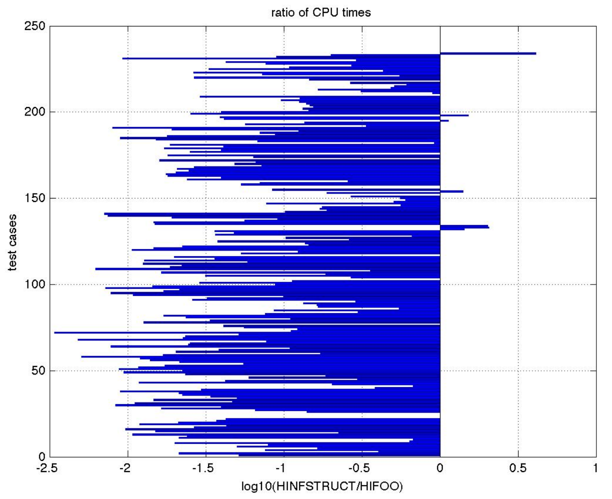


Fig. 6. Comparison of achieved objectives (top) and CPU times (bottom)

[3] D. McFarlane and K. Glover, "A loop shaping design procedure using H_∞ synthesis," *IEEE Trans. Aut. Control*, vol. 37, no. 6, pp. 759–769, 1992.

[4] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*. Prentice Hall, 1996.

[5] P. Gahinet and P. Apkarian, "A linear matrix inequality approach to H_∞ control," *Int. J. Robust and Nonlinear Control*, vol. 4, pp. 421–448, 1994.

[6] *Robust Control Toolbox 4.1*. The MathWorks, Inc., Natick, MA, USA, March 2012.

[7] P. Apkarian and D. Noll, "Nonsmooth H_∞ synthesis," *IEEE Trans. Aut. Control*, vol. 51, no. 1, pp. 71–86, 2006.

[8] P. Apkarian, V. Bompart, and D. Noll, "Nonsmooth structured control design with application to PID loop-shaping of a process," *Int. J. Robust and Nonlinear Control*, vol. 17, no. 14, pp. 1320–1342, 2007.

[9] R. M. Redheffer, "On a certain linear fractional transformation," *J. Math. and Phys.*, vol. 39, pp. 269–286, 1960.

[10] J. Doyle, A. Packard, and K. Zhou, "Review of LFT's, LMI's and μ," in *Proc. IEEE Conf. on Decision and Control*, vol. 2, Brighton, Dec. 1991, pp. 1227–1232.

[11] A. Varga and G. Looye, "Symbolic and numerical software tools for LFT-based low order uncertainty modeling," in *Proc. CACSD'99 Symposium, Cohala*, 1999, pp. 1–6.

[12] A. Varga and J. Magni, "Enhanced LFR-toolbox for MATLAB," *Aerospace Science and Technology*, vol. 9, no. 2, pp. 173–180, 2005.

[13] C. J. Bett and M. Lemmon, "On linear fractional representations of multidimensional rational matrix functions," Eindhoven, University of Technology, Tech. Rep., 1997.

[14] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control - Analysis and Design*. Wiley, 1996.

[15] J. V. Burke, D. Henrion, A. S. Lewis, and M. L. Overton, "HIFOO - a MATLAB package for fixed-order controller design and H_∞ optimization," in *5th IFAC Symposium on Robust Control Design*, Toulouse, France, July 2006.

[16] S. Gumussoy, M. Millstone, and M. L. Overton, "H_∞ strong stabilization via HIFOO, a package for fixed-order controller design," in *Proc. IEEE Conf. on Decision and Control*, Cancun, Mexico, 2008, pp. 4135–4140.

[17] P. Apkarian and D. Noll, "Nonsmooth optimization for multiband frequency domain control design," *Automatica*, vol. 43, no. 4, pp. 724–731, April 2007.

[18] P. Gahinet and P. Apkarian, "Structured H_∞ synthesis in MATLAB," in *Proc. IFAC*, Milan, Italy, Aug. 2011.

[19] P. Apkarian, "Internet pages," http://pierre.apkarian.free.fr, 2010.

[20] J. Blight, R. Dailey, and D. Gangsassi, "Practical control law design for aircraft using multivariable techniques," *International Journal of Control*, vol. 59, no. 1, pp. 93–137, 1994.

[21] C. Luo, R. Liu, C. Yiang, and Y. Chang, "H_∞ control design with robust flying quality," *Aerospace Science & Technology*, vol. 7, pp. 159–169, 2003.

[22] F. Leibfritz, "COMPL_eIB, COnstraint Matrix-optimization Problem LIbrary - a collection of test examples for nonlinear semidefinite programs, control system design and related problems," Universität Trier, Tech. Rep., 2003.