DSP FUN THE GUI WAY

A. Yardim , M.A. Mughal , G.D. Cain and D. Barjamovic

University of Westminster, Department of Electronic Systems, London W1M 8JS, UK

ABSTRACT

Production of Graphical User Interfaces (GUIs) has been found to be a vital ingredient in building up a comprehensive MATLAB framework supporting the learning - and enjoyment - of DSP topics. We describe some of the functional and esthetic issues surrounding the two dozen GUIs we have developed and used to animate various DSP topics during in-class demonstrations and in student laboratories. We conclude that the substantial development effort needed for a really good GUI is paid back by the impact on learning and the enhancement of routine design and measurement tasks.

MATLAB is so useful, in so many ways, across a broad range of DSP learning situations that it is easy to highlight many of its strengths. Apart from widespread use by our researchers as their everyday concept-proving tool of choice, our main interests lie in undergraduate learning (immediately upon entry to Year 1 of study and then throughout three years of BEng work), in Master's study, and also in Continuing Professional Development short courses aimed at practicing Engineers. MATLAB is an indispensable accelerator of learning and engine of productivity enhancement at each of these rungs of our educational ladder.

It is fortunate for DSP aficionados that our subject is inherently SENSUAL, and that we can so readily convey high-impact messages about the intricacies of its various sub-topics by appealing to a student's senses of sound and vision. A noise-contaminated sound passage played out loud is guaranteed to register its undesirability more compellingly than a dry signal-to-noise equation or stark graph. Even better, the collision and adhesion of two tones leaping about in an animated spectral display evokes dynamism and excitement on the part of a fledgling DSP engineer charged with the task of unraveling these interfering signals. MATLAB supplies a framework for readily enlivening these and classroom demonstration and laboratory scenarios so powerfully that modern teaching of DSP need no longer suffer under the dry mathematical image which used to unsettle newcomers to the field.

We try to insist that the learning of DSP be made FUN. In pursuit of this, we use MATLAB in these various classroom roles:

- "Scratch-pad" trial of concepts on the fly
- "Pause Stories" for automated, pre-canned expositions (in a "slide show" vein)
- "Sound-file hacking" to gain a quick feel for processing effects
- M-file creation for small task solution
- Serious m-file utility construction to augment Toolbox features
- "Control Panel" GUI tool creation for repetitive use

These usage styles are in increasing order of sophistication and investment on the part of academic and support staff. The first two are inexpensive, but often add spontaneity along structured lines of exposition. Things start getting especially interesting with "sound-file hacking". This code fragment:

```
[x,fs]=wavread('terminat.wav');
y=[x;flipud(x)];plot(y);sound(y,fs)
```

is easy for students in a PC-equipped lecture room to quickly input and grasp, yet never fails to deliver a few warm chuckles of amusement. The final three uses incur escalating effort and seriousness, but deliver far greater learning benefits. The developer in each case might also be the student (and certainly in the homework and project assignment situation that often happens); more often, dedicated academic effort behind the scenes has been deployed and the student is in the happy position of simply "riding" on what gets provided. Slick, labour-saving specialist m-files can be pressed into service so that students can focus on grasping the concepts, not stumbling over the tools.

Notwithstanding the best efforts of the development team (building on the great transparency and self-documenting simplicity MATLAB brings to the party anyway), we have frequently been astonished to find very, very slow rates of code development in abbreviated in-class jobs or even 3-hour laboratory sessions. Some students simply cannot amalgamate high-level concepts and simple coding to do useful work under pressure of time. Acceleration of signal handling is a *must*. Hence our enthusiasm for Graphical User Interfaces (GUIs).

GUIs are great for setpiece, repetitive tasks. We like to view them as low-cost items of dedicated, powerful DSP instrumentation: "control panels" for unleashing magic. Our GUI production efforts have been directed both at in-class demonstration of concepts for making lectures lively and at close support of design and measurement. The first usage has been effective where complex signals have to be visualized, where z-plane topology needs to be seen, where pole/zero geometries require crystallization, where surveying filterbank outputs can illuminate the signal processing landscape, etc.

We find it handy to have a small selector toolbar visible for easily invoking the couple of dozen GUIs we tend to use routinely. This arrangement (seen in the right corner of Figure 1) gives both an alphabetic ordering and visual cues to the lecturer (who may be hard-pressed to recall the name of the intended demo). Figure 1 also exhibits the result of a couple of selections; partially covered is a 3-d "corkscrew" depiction of complex exponential signals which has been immensely helpful in asserting the reality of our most important complex signals. Also in Figure 1 is *slifer* - one of our most popular tools for handcrafted digital filter design, permitting highly responsive adjustment of individual coefficient values (here of a digital differentiator), variation of Frequency Sampling transfer function values and even non-equispaced spectral manipulation which MATLAB's fast handling of Vandermonde matrix inversion causes to look effortless.

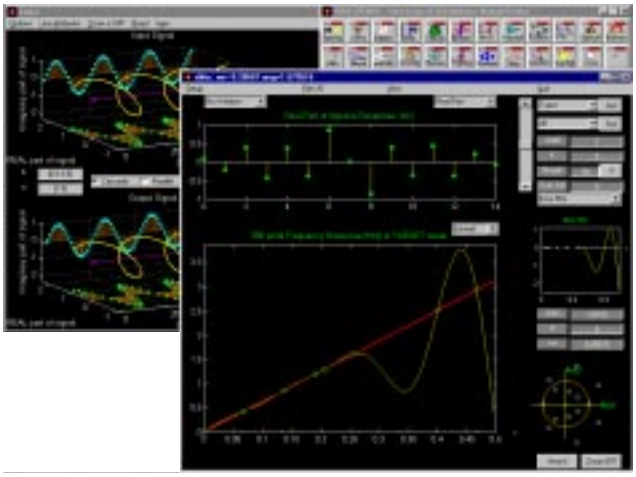

Figure 1. A Small Collage of Our GUIs

Figure 2 shows a view of the z-plane rarely seen by students elsewhere. Here we have taken the usual MATLAB coefficient vectors to be **a=[1 0.6**] and **b=ones(1, 5)**. The student is able to horizontally slice the z-function finely enough to scrutinize the surface perturbation caused by the pole on the negative real z-axis while also seeing the DFT evaluations shown around the unit circle by a stemplot. As different pole and zero contributions are imported the student is able to observe the undulations caused in the z surface and how these are manifested "at the edge", where the DFT of the transfer function resides. Such potent and usable tools inspire both confidence

in the theoretical ideas and deliver practical outcomes for meeting tough design specs. Students react extremely well to the solidity and reliability that tools such as these represent.

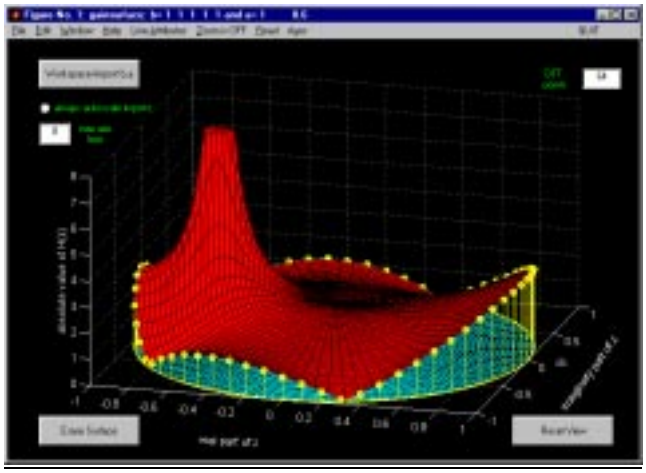

Figure 2. Circular-Slicing a Z-Transform

But can we expect students to design good GUIs? Certainly such an aspiration is smack on target for modern groupwork-intensive, creative student-led engineering learning. Yet our experience is mixed; several individual projects have produced superb, highly useful GUIs that have gone on to become permanent fixtures in our toolscape. But many have been feeble, painful exercises too. Far too often students have difficulty articulating and then implementing features which are truly useful. The whole process is complicated by the inherent difficulty of structuring a GUI. The exposure to switchyard programming is very educational, but the mechanics of callbacks and ponderous syntax are not easy to master for beginners. Brave attempts like MATHWORKS' "Guide" are not yet able to simplify development sufficiently. At the present time, you have to really want to build a GUI if it's going to be much of a success.

In developing our GUI outlook for learning aids, we were greatly influenced by MATLAB's demo *sigdemo2*, which is wonderfully economical, self-explanatory and useful. This one example immediately suggests a host of transform illustrators that could provide good insight for students; no longer is it necessary for transform tabulations to be so dry and uninspiring. We created a "Hilbert Transform Tour" which both draws together a number of these elusive, unruly transform pairs (as time-domain equations), and also animates them so that the browsing student can, under slider control, easily modify signal parameters and appreciate changing patterns and interactions (especially as manifested in analytic signals).

Filter design also furnishes a wealth of opportunities for animation, display and measurement GUIs. MATLAB's *filtdemo* is a (hard-hitting) case in point. We have sometimes found it useful to "soup up" such nice demos through small modifications that add sensual impact (say, by incorporating music processing). And we can go deep, to inspect the iteration-by-iteration mechanics of filter design algorithms. Such iterative minutiae can be fascinating to watch, while also leaving the student with a feeling of the precariousness of iteration (especially when convergence fails to happen) and the potency of closed-form solutions and matrix formulations.

Many instrumentation tasks cry out for purpose-built GUIs. This is especially true in the area of random signal measurement, an arena notorious for requiring gigantic record length/ensemble sizes (tens to hundreds of thousands of samples) before any textbook findings can be validated convincingly. Here repetitive mechanization (and some sort of averaging) of random signal realizations is the focus. So far we have attacked this topic only by m-file utilities; it is one of our next targets for GUI development.

What features should a good GUI exhibit? We think that the envelope of functionality should be restricted, and obvious to the user. Excessive comprehensiveness leads to a lumbering GUI that is neither fish nor fowl (and greatly extends its development schedule). Its controls and layout should be friendly and inviting, with balance and harmony uppermost. Spaciousness

of uicontrol object deployment is always hard to achieve, and may require deeper "parking" (into menu items, dialog boxes or popup menus) than the GUI developer would like. The user should only rarely be called on to have to push further down than 3 levels of interaction hierarchy (i.e., 2 button clicks).

A user should feel the power of the GUI's operation, and breathe a sigh of relief at not having to get down to the nitty-gritty of doing the main underlying coding personally. Any GUI that's hard to learn to drive will simply be abandoned in favour of the ease of DIY that MATLAB always holds out to users (particularly those who have built up a potent and familiar suite of specialist m-file utilities). Any GUI that is mysterious, illogical, unreliable and burdensome to navigate will incur needless Operator Fatigue and will breed not just disdain- but active hatred. Again, such a failed GUI is headed for the scrap heap.

We think that GUIs come in "three time flavours": those that are developed in 3 minutes, 3 days or 3 months. The 3-minute quickie can only be something simple like adding a slider to figure window to facilitate observation of parameter change effects, adding a pushbutton to invoke sounds or flash colour changes, and so forth. Such limited aspirations almost always pay back the effort adequately. At the other extreme, really ambitious and polished GUI goals can easily take 3 months or more. Our experience is that these very comprehensive GUIs are daunting even for veteran GUImakers and degenerate into intricate interactions of control conditions that can be hard to justify consistently, much less to remember for maintenance and upgrading.

Our favourite category is the 3-day type of GUI. Here the mindset is limited to specific horizons which often prove achievable. Our best GUIs are up and running (as regards their essential features) very rapidly. Subsequent fine-tuning and considered extension make go on over many months, of course, but the fact remains that it was a 3-day concept, it gave at least a limited degree of service in about 3 days, and it benefited from very early user feedback. Throughout the process, sketching - and re-sketching - the control panel layout before investing in coding proves to be a key factor in homing in to a good GUI.

Figure 3 shows *demoleak* - the first GUI to emerge from our group. This provided our first view of the dynamic effects of leakage in spectral analysis as a tone was forced off the DFT gridpoints, and still provides impressive testimony to the potency of windowing.

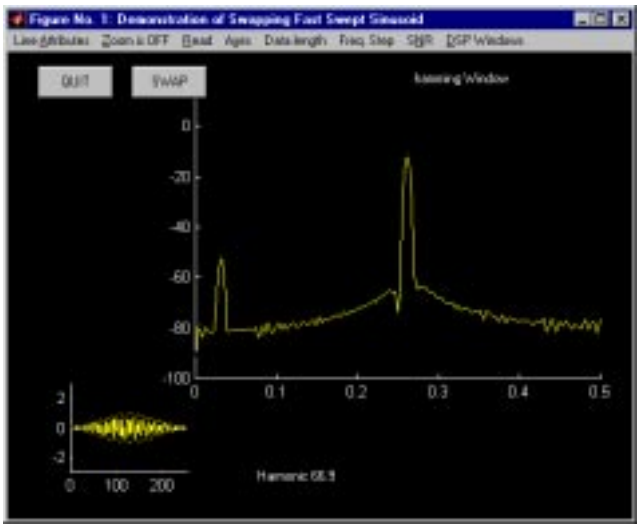

Figure 3. A Dynamic Spectral Analysis GUI

Where then is the "fun" in all this? There is no doubt that it is the end users (and not the harassed GUI developers) which stand to have the bulk of the fun. They have but to test drive the final product. Our final judgement is unequivocal: a good GUI is a powerful aid in breaking down resistance to the mathematical matters which underpin DSP. A reasonably good GUI is pretty easy to achieve; a great GUI is rare, but propels its user to a state of DSP-joy.

ACKNOWLEDGEMENT

We are grateful for the good programming style and creative energy that Dr. Greg Allen of James Cook University - father of *demoleak* and much more - brought to our GUI factory.