equation—an approach that applies to a larger set of dc-dc converters. Although it does not have a closed form convenient for analog circuits, storing the surface as pixels in a raster image on embedded processors, using the techniques described, is both accurate and computationally efficient.

A common misconception about geometric control is that parameter-dependent switching surfaces are not robust. Linear SMPS controllers usually deal with parameter uncertainty by using error integrators, components often excluded in geometric controllers. It was shown that, even with some parameter uncertainty, geometric control (without error integrators) outperforms many conventional SMPS controllers (also without error integrators) for many common large-signal disturbances. Section 3.5 investigates online system identification as a way to improve parameter knowledge.

3.5 Parameter Observers

To calculate a surface that minimizes response time, MTC requires all the parameter and state knowledge inputs, knowledge a controller may not have available. State and parameter observers are two techniques that increase controller knowledge. State observers are a well established technique for obtaining state knowledge in the absence of sensors. Parameter observers [76] are a less common technique that identifies parameter change, the root cause of voltage transients. This section investigates system identification to calculate power converter parameters via simulated and hardware experiments.

From a system perspective, the two observers serve very different functions. For example, an SMPS controller can either estimate voltage and current measurements by assuming fixed parameters or it can estimate parameters by measuring voltage and

current. Because today's digital controllers can make numerous measurements with multiplexed a/d converters, parameter observers are a logical next step for the industry.

Models and simulation are offline tools that aid engineers in selecting feedback gains for a static operation point. Often a designer does not have access to an accurate plant model, due to its sheer complexity or nondeterministic plant changes. The latter case commonly occurs in dc-dc converters. With online system identification, real-time algorithms autonomously monitor plant parameters, such as SMPS loads. The more up-to-date parameter knowledge a controller has, the better it can select gains and predict. The following sections categorize knowledge accessible to control and the ways to store this knowledge.

3.5.1 A priori and posteriori knowledge

A priori encompasses all knowledge before an applied input. This could include static parameters such as source voltage, inductance, switching frequency, etc. It can also include nonstatic parameters such as periodic load changes. A designer can choose feedback gains based upon an assumed model and previous measurements or directly from a previous frequency response analysis (FRA). Two issues exist with these gain selection options. Models are no substitute for physical measurements, and FRA methods with frequency sweeps are both time-consuming and valid only for a single parameter set. Ideally, a controller should detect load changes and revise its control law online. This is akin to "adaptive," "gain scheduling," "self-tuning," and "optimal" controls. These methods have a long history in the process control industry [77] and potential use in dc-dc power converters. SMPS controllers that can quickly identify their loads can more accurately place poles [78, 79], better enable transient-suppressing circuits [80-82], and

calculate optimal switching trajectory paths [60, 67]. System identification, the overlooked core of these advanced control methods, learns about the plant rather than assume a priori values.

Posteriori classifies all information obtained through direct input and output observations using system identification algorithms. From previous inputs, outputs, and deterministic perturbation sources, algorithms can estimate parameters for frequency and time-domain models. Classic schemes inject noise (ideally white) in a way that will not drastically affect the output performance and then record the results. Noise-based algorithms typically have smaller identification windows than frequency-sweep algorithms. The former speed up identification times but can only describe local model behavior. Texts distinguish algorithms as either *nonparametric* or *parametric system* identification [2, 83, 84].

3.5.2 Nonparametric versus parametric models

The more popular *nonparametric* plant identification has an undefined model structure and parameter vector size—a key advantage in difficult-to-model SMPSs. It usually displays results as frequency and phase diagrams. It gets its name from the fact that no finite-dimensional parameter vector can store this model, a trait explained through the following generalized scenario.

Suppose a constant coefficient difference equation (CCDE)

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{m=0}^{M} b_m u[n-m], \tag{31}$$

with discrete inputs $u[n] = u(nT)$, outputs $y[n] = y(nT)$, and sample time T, perfectly models a linear time invariant (LTI) plant with finite-dimensional plant parameters (i.e.,

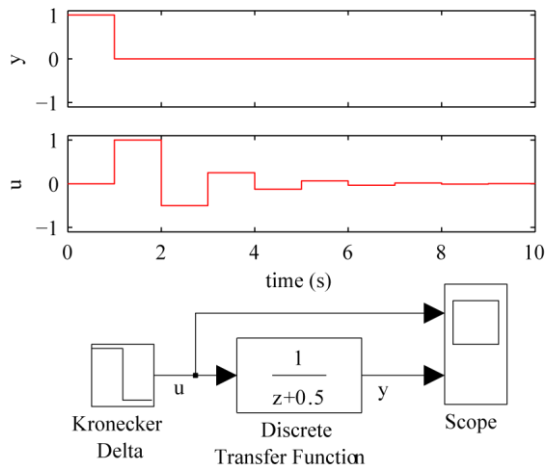
it has a parametric model). When the plant is injected with a unit impulse, and $N \geq 1$, the output has an infinite-duration impulse response that defines a plant impulse response $h[n]$. If the plant structure is known a priori, one can solve $h[n]$ directly over a *finite-time* duration using

$$y[n] = h[n] = -\sum_{k=1}^{N-1} \frac{a_k}{a_0} y[n-k] + \frac{b_n}{a_0} \tag{32}$$

for $0 \leq n \leq M$. Now consider an LTI plant with an unknown structure. The unit impulse response still defines $h[n]$, but one needs an *infinite-time* duration to solve for $h[n]$, as demonstrated by an LTI system's unit-impulse response

$$\{y[n] = h[n]\} \xrightarrow{\mathbb{Z}} \left\{Y(z) = H(z) = \sum_{n=0}^{\infty} y[n] z^{-n}\right\}. \tag{33}$$

Differentiation between parametric and nonparametric discrete transfer function models is further illustrated within Fig. 40. The equations show that a plant with a known



Parametric model: $H(z) = \frac{1}{z+0.5}$

Nonparametric model: $H(z) = 1z^{-1} - 0.5z^{-2} + 0.25z^{-3} \ldots$

Fig. 40 Parametric and nonparametric model example

model structure can identify itself faster with parameter rather than nonparametric algorithms.

Parametric algorithms identify plants according to candidate models over a finite sampling window—a key speed advantage for simple plants. These models are further classified as having either gray- or black-box model structures [83, 85]. *Black-box* models treat parameter coefficients purely as numbers and assign arbitrary length parameter vectors, while *gray-box* models have coefficients with physical meaning (such as resistance, inductance, capacitance, etc.) and have a parameter length tailored for the plant. A priori component values that define initial gray-box model coefficients help parametric identification algorithms converge faster and can serve as a base comparison for the identified plant.

While parametric identification works well for simple-to-model plants, circuits can exhibit nonlinear behavior that CCDE models cannot describe. Here, engineers use phase- and frequency-response models to design compensators. Their use justifies the popularity of nonparametric methods for power electronics. Posteriori nonparametric system identification with cross-covariance methods has been studied for forward converters in [85, 86] where the authors present output voltage-to-duty cycle-frequency responses. The forward converter with an ideal transformer has a simple large-signal model described with a CCDE. Small-signal system identification algorithms with black-box parametric models have been studied for the forward converters [85] and in general for buck, boost, and buck-boost converters [87]. This dissertation investigates large-signal identification for synchronous buck converters with a gray-box parametric model. This alternate perspective leads to the following analyses:

- Identified and model parameter coefficients are compared via time-domain algorithm metrics: prediction error, parameter error and convergence time.
- Hardware and simulated experiments show accuracy and speed of algorithm estimates after an abrupt load change.

Section 3.5.3 presents a candidate model for a continuous-time synchronous buck converter and Section 3.5.4 transforms it into a discrete-time model for digital implementation. Finally, Section 3.5.5 reviews the system identification algorithm called *recursive least squares* (RLS) and tests it via simulation in Section 3.5.7 and hardware in Section 3.5.8.

3.5.3 Candidate power converter model

The candidate model defines how a system identification algorithm maps samples into parameter estimates. It should approximate plant dynamics and update after each sample. A likely model structure, (31) is often classified as an autoregessive-moving average or Box-Jenkins model, which models discrete LTI systems. The synchronous buck modeled in Section 2.1 was an LTI model, but it must be in discrete-time CCDE in order to run as a recursive algorithm.

The SSA buck converter model, presented in Section 2.1, forms two single-input single-output, large-signal transfer functions, whose calculation was shown in Appendix A.1. The output inductor current-to-input duty, $H_I(s)$, and the output capacitor voltage-to-input duty, $H_V(s)$, are

$$\begin{bmatrix} I_L(s) \\ V_c(s) \end{bmatrix} = \begin{bmatrix} H_I(s) \\ H_V(s) \end{bmatrix} U(s)$$

$$= \begin{bmatrix} \dfrac{E_1 C_1 (R_1 r_{C1}) s + E_1}{L_1 C_1 (R_1 + r_{C1}) s^2 + s(L_1 + C_1 R_1 r_{L1} + C_1 R_1 r C_1 + C_1 r_{L1} r_{C1}) s + (R_1 + r_{L1})} \\ \dfrac{E_1 R_1}{L_1 C_1 (R_1 + r_{C1}) s^2 + (L_1 + C_1 R_1 r_{L1} + C_1 R_1 r C_1 + C_1 r_{L1} r_{C1}) s + (R_1 + r_{L1})} \end{bmatrix} U(s). \quad (34)$$

The equivalent lumped parameter transfer function is

$$\begin{bmatrix} H_I(s) \\ H_V(s) \end{bmatrix} = \begin{bmatrix} \dfrac{b_0 s + b_1}{s^2 + a_0 s + a_1} \\ \dfrac{b_3}{s^2 + a_2 s + a_3} \end{bmatrix}. \quad (35)$$

The state-to-duty cycle transfer functions, $H_V(s)$ and $H_I(s)$, have identical pole structures. However, the zero in $H_I(s)$ has a pole-canceling effect that gives it a slightly faster dynamic response. When the duty cycle changes, the capacitor voltage lags behind the inductor current. This attribute makes $H_I(s)$ a better candidate model than $H_V(s)$ for fast system identification. A disadvantage with identifying with $H_I(s)$ is the increased measurement noise associated with usually higher current ripple. Provided that the converter operates at fast switching frequencies, it will still be accurate, as was previously shown in Fig. 3. The model must also be converted into discrete-time form in order to correlate it to digital samples.

3.5.4 Discrete-time model

Most system identification algorithms perform math on samples, a practical task for digital processors. Sampling adds signal distortion in the form of quantization error, delay, and a transfer-function pole. A system identification algorithm must account for

these effects in its discrete-time model. The symbolic mapping from discrete-time coefficients to continuous-time coefficients allows one to solve for the physical component values (provided some are known a priori). The symbolic mapping from *a* and *b* coefficients to circuit parameters is shown in Appendix D.1. Unfortunately, when the continuous-time models are converted into discrete-time models, the mapping for *c* and *d* coefficients has excessive term counts. This problem was circumvented by lumping circuit parameters using their numerical values. The inductor current discrete-time model, $H_I(z)$, was derived by lumping $H_I(s)$ parameters and applying the ZOH approximation. This discrete-time model has the general tractable form

$$H_I(z) = \frac{d_1 z^{-1} + d_2 z^{-2}}{1 + c_1 z^{-1} + c_2 z^{-2}} = \frac{Y(z)}{U(z)}, \tag{36}$$

where the parameter coefficient vectors, *c* and *d*, depend on the *a* and *b* coefficients and on sample time *T*. The simple transfer function still describes a dynamic system, but insight on how circuit components contribute to its response is lost. System identification algorithms estimate these discrete-time numeric coefficients using previously sampled inputs and outputs. The RLS algorithm reviewed in Section 3.5.5 was chosen for this task.

3.5.5 Review of recursive least square algorithm

Least-squares methods solve for a model that minimizes output error in a sample set, similarly to how curve fitting finds the closest polynomial coefficients that match a measurement set. The same concept applies to sampled data systems where an estimated transfer function

$$H(z) = \frac{d_1 z^{-1} + d_2 z^{-2} + \cdots d_M z^{-M}}{1 + c_1 z^{-1} + \cdots c_N z^{-N}}, \tag{37}$$

must closely match *M* previous inputs and *N* previous outputs. System identification algorithms performed on digital processors have memory that stores previous samples and arithmetic logic units that correlate the inputs and output into plant estimates. The sampled output in terms of previous inputs and outputs is

$$y[\mathrm{k}] = \begin{Bmatrix} \mathrm{y}[\mathrm{k}-1] \\ \mathrm{y}[\mathrm{k}-2] \\ \vdots \\ \mathrm{y}[\mathrm{k}-\mathrm{n}] \\ \mathrm{u}[\mathrm{k}-1] \\ \mathrm{u}[\mathrm{k}-2] \\ \vdots \\ \mathrm{u}[\mathrm{k}-\mathrm{m}] \end{Bmatrix}^{\mathrm{T}} \begin{Bmatrix} -c_1 \\ -c_2 \\ \vdots \\ -c_N \\ \mathrm{d}_1 \\ \mathrm{d}_2 \\ \vdots \\ \mathrm{d}_M \end{Bmatrix} = \Phi[\mathrm{k}-1]^{\mathrm{T}}\theta_0. \tag{38}$$

The *regression vector*, $\Phi[k-1]$, contains all previous input and output measurements, and θ_0 contains the actual system parameters. The candidate model, processing, and memory constraints limit regression length, so the algorithm must calculate a finite set of estimated parameters $\hat{\theta}$ and outputs $\hat{y}[k]$. The RLS algorithm calculates the $\hat{\theta}$ that minimizes the sum of errors:

$$\min_{e[k]} e[k] = \sum_{r=1}^{k} \left(y[r] - \Phi[\mathrm{r}-1]^{\mathrm{T}}\hat{\theta}\right)^2. \tag{39}$$

Solving (39) for $\hat{\theta}$ and expanding summing terms transforms the result into the well known RLS form

$$\begin{gathered} \hat{\theta}[k] = \hat{\theta}[k-1] + R[k-1]^{-1}\Phi[\mathrm{k}-1]\left(\mathrm{y}[\mathrm{k}] - \Phi[\mathrm{k}-1]^{\mathrm{T}}\hat{\theta}[k-1]\right) \\ R[k] = R[k-1] + \Phi[k]\Phi[k]^{\mathrm{T}}, \end{gathered} \tag{40}$$

where $R[k]$ represents the autocorrelation matrix. Equation (40) is a special case of Kalman filter. RLS has other solutions that do not invert matrices, as well as adjustable *forgetting factors* [83] that weigh recent samples more than older ones. These solutions

improve computation and tune identification performance, respectively. The RLS algorithm implemented in experiments did not invert matrices. It weighted samples uniformly and considered sensor scaling. The derivation is found in Appendix D.2. This dissertation uses the RLS algorithm on a synchronous buck converter. Here, output $y[k]$ is the sampled inductor current, $i_L(t)$, and $u[k]$ is the sampled PWM duty cycle. Section 3.5.6 describes metrics used to assess SMPS system identification performance in simulation and hardware.

3.5.6 System identification performance metrics

System identification results will be measured in terms of accuracy and convergence time. *Convergence time* is the length of time, τ, for a controller to reach some small error ϵ. An *accurate identification* has the property $\left|\hat{\theta}[k] - \theta_0\right| < \epsilon_\theta$ where ϵ_θ, the paramaeter error, is small for $k > \tau$. *Accurate prediction* has the property $|\hat{y}[k] - y[k]| < \epsilon_y$ where ϵ_y, the prediction error, is small for $k > \tau$. The *parameter error norm* is another expression that lumps multiple parameter errors into a single quantity via the two-norm. Adaptive and predictive SMPS controllers need fast convergence time so that gains update immediately after detected load steps. The RLS algorithm was chosen specifically for its desirable performance after abrupt parameter changes [88]. It is important to recognize that RLS system identification is an empirical method which fits data to an estimated plant. The reverse process is seen more commonly in publications, where a model derived from first principles must match measured data. Section 3.5.7 presents the simulated results—an ideal model and measurement scenario—and Section 3.5.8 presents the hardware results—a less than ideal scenario. Simulations

demonstrate algorithm proof of concept, while the hardware introduces nonlinear components, system and sampling noise, and computation practices.

System-identification algorithm performance will depend on input excitation, initial conditions, and model/sample error. Known-input excitation produces correlated output change that helps describe the unknown plant. Persistent excitation keeps the plant estimate up to date during a steady-state output. For example, the synchronous buck converter model in has a steady-state voltage output, $v_C = E_1 u$, when $R_1 \gg R_{L1}$. Since its steady-state output does not depend strongly on R_1, direct v_C measurements with a constant input duty cycle cannot provide samples that correlate to the load. However, fast input changes expose the plant dynamic behavior at the output, which does depend on components such as R_1—the higher the input spectrum density, the better the system identification performance. Three different input excitations are explored in this dissertation: step impulse, white, and pink noise. Initial conditions also affect identification performance. Poorly chosen initial conditions extend convergence time or create algorithm instabilities. Therefore, initial values were chosen such that RLS variables were bounded during the algorithm startup. All experiments assigned $\Phi[0]$ to the null vector, $\hat{\theta}[0]$ to values greater than ±50% of model plant parameters, and $R[0]$ to the identity matrix.

Model and rounding noise are unavoidable artifacts caused by unmodeled behaviors. Noise increases the minimum mean-square prediction error. For example, model (36) is an SSA, which does not contain switching dynamics. Thus, effects such as ripple and equivalent-series-resistance jumps will distort signals and contribute as

nondeterministic noise sources. More rigorous descriptions on how quantization and measurement noise affect the RLS algorithm are described in [89].

Rounding noise comes from quantization on the sensors, d/a converters, and from finite numeric precision. The simulation and hardware operated on unscaled input and output integers between 0-4095 and 0-1250, the ranges for the DSP a/d converters and PWM modules, respectively. Digital signals, $u[k]$ and $i_L[k]$, are plotted in this raw integer format. The a/d scale factor for $i_L(t)$ was 2.38 mA per bit, and for $u(t)$, 0.08 % duty cycle per bit. The RLS without matrix inversion was computed using 32-bit floating-point numbers. The author attempted to make a fixed-point version for the algorithm to take advantage of the Texas Instruments TMS320F2812 DSP math hardware. Although $R[k]$'s large numeric range produced many challenges, even without much code optimization, the algorithm achieved a 4 kHz maximum control-loop rate. Faster performance is possible on a DSP or an FPGA with an FPU. The following simulation accounts for the mentioned round-off noise and maximum control-loop speed.

3.5.7 Simulated results

Simulation served as a controlled environment for perfecting the rather complicated RLS algorithm. Here, we could more easily observe round-off and data-type effects and generate repeatable noise sequences. The simulation model was PWL with parameters, $R_1 = 5\ \Omega,\ L_1 = 1.26$ mH, $R_{L1} = 150$ mΩ, $C_1 = 1.29$ mF, $R_{C1} = 5$ mΩ, and $E_1 = 10.0\ V$, taken either from manufacturer-specified values or from LCR meter measurements. The converter switched at 60 kHz and sampled $i_L(t)$ every 250 μs to allow time for RLS computations. The converter was injected with a 50% duty-cycle step input superimposed on 0.2 μW/Hz white noise at 50 ms, as shown in Fig. 41. At 200 ms,

R_1 changed from 5 Ω to 1 Ω and the algorithm updated. Before the load step, the candidate model was

$$H_I(z) = \frac{I_L(z)}{U(z)} = \frac{1.935z^{-1} - 1.862z^{-2}}{1 - 1.895z^{-1} + 0.9328z^{-2}}. \tag{41}$$

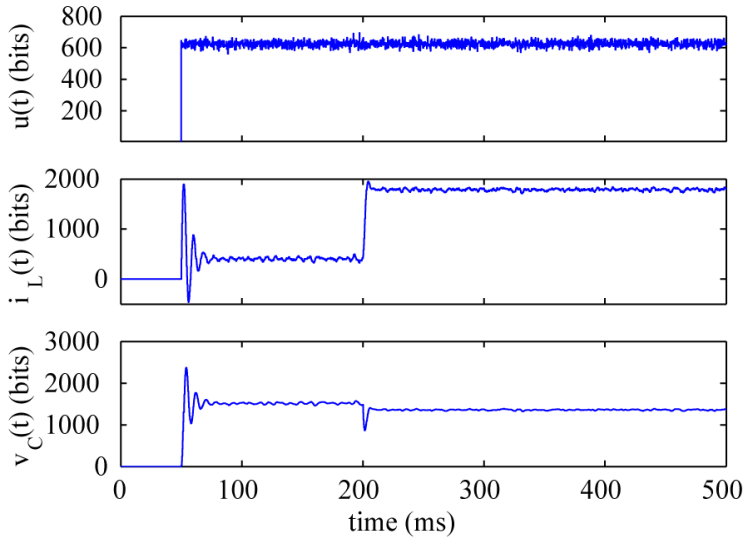


Fig. 41 Simulated digital input and output signals

The injected step input produced a rich response at the output, a property that improves system identification performance. Converter startup provided a unique system identification opportunity. After only 2 ms the algorithm estimated a plant with a less than ±6% parameter error, as shown in Fig. 42 and Fig. 43. This convergence time was less than the 4 ms it takes for the output to reach its peak voltage—a result with two key control implications. First, the inductor current identification scheme learned the open-loop transfer function coefficients, a necessary step in adaptive pole-placement. Second, these coefficients were learned before output reached its peak value; i.e., control can predict all future trajectories to reach the new equilibrium. The predication and parameter error norms are shown in Fig. 43. Predication error is a one-sample period future-output prediction. Its value stayed below ±10% through the simulation, indicating a practical use

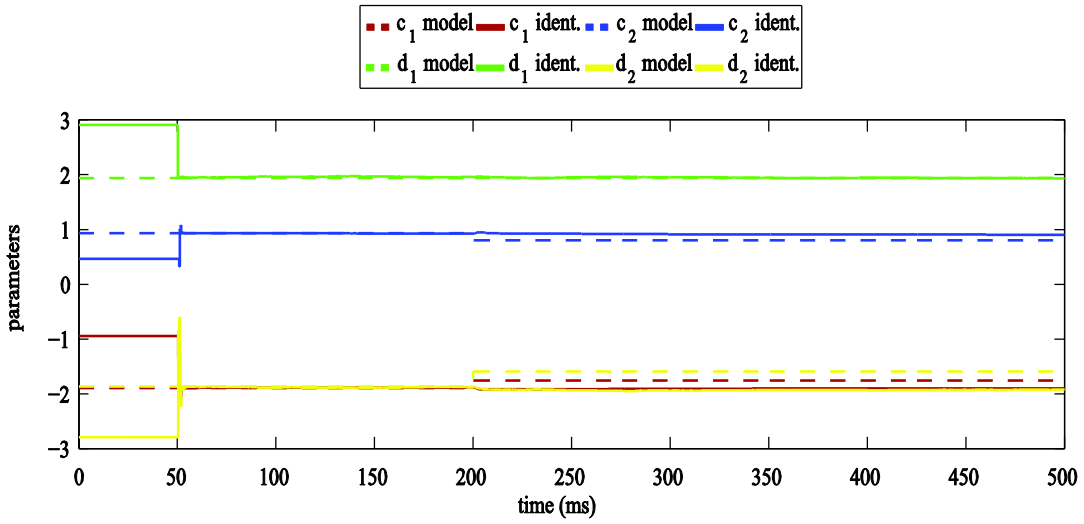


Fig. 42 Simulated parameter estimate vector

for dead-beat control. When the load step occurred, output was correlated to the white-noise input alone. The white-noise input did not produce as rich an output as that seen with the step input. Post-load-step coefficient estimates, d_2 and c_2, had 20% and 13% parameter error, respectively. A ten-fold increase in switching and sampling frequency could not elevate this error.

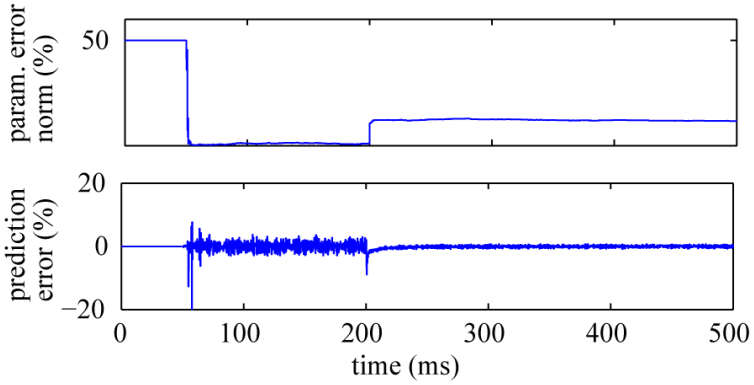


Fig. 43 Simulated system identification performance

The simulation was an important tool that helped translate concepts into embedded code and described expected measurements for the 50 W hardware buck converter that will be described in Section 3.5.8. Simulations showed RLS could create an accurate

model during startup, but it loses accuracy when steady-state plant parameters change abruptly.

3.5.8 Hardware experiment

The custom RLS algorithm was programmed using embedded Matlab™, a subset language, that readily translates M code into C code intended for embedded targets. The embedded function is listed in Appendix D.3. The software setup is shown in Fig. 44. A 200 ms noise sequences was generated off line, sampled at 4 kHz, uploaded into DSP memory, and added to $u[k]$. For experiments that lasted longer than 200 ms, the noise sequence repeats. The sampled and calculated variables, $i_L[k]$, $\widehat{i_L}[k]$, and $\hat{\phi}[k]$, were downloaded using Link-for-Code Composer Studio™. The downloaded data was solely for post-experiment analysis—all RLS estimates were formulated on the DSP in real-time.

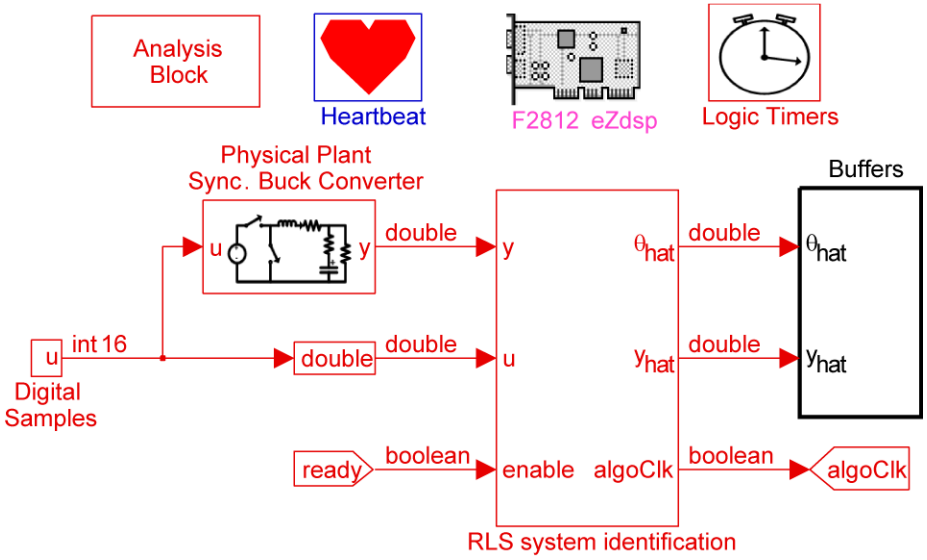


Fig. 44 Simulink simulation and embedded DSP code

Two different hardware experiments were performed. The first was a short-term, high-fidelity experiment where samples and algorithm variables were buffered over a 200

ms period. The synchronous buck converter powered a 5 Ω load and superimposed white noise on a 50% duty-cycle step input. Three analog measurements are shown in Fig. 45. The edge-to-edge clock in *CH3* measured the control-loop speed. The drain-to-source voltage in *CH2* shows the plant input, and the *CH4* output capacitor voltage shows power quality in the presence of white noise.

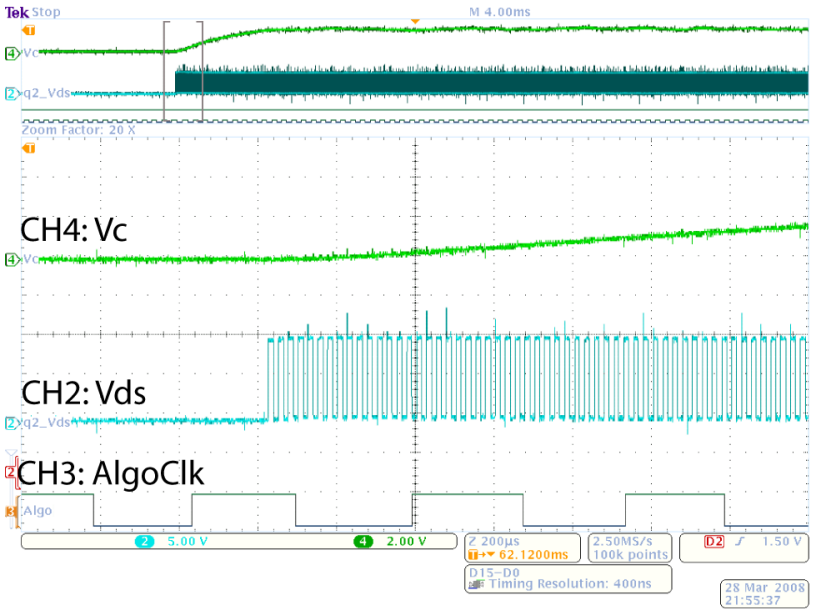


Fig. 45 Analog measurements (*CH4* capacitor voltage, *CH2* voltage across q_2, and *CH3* algorithm clock)

In Fig. 46 measured a/d signals are compared to those expected from simulating

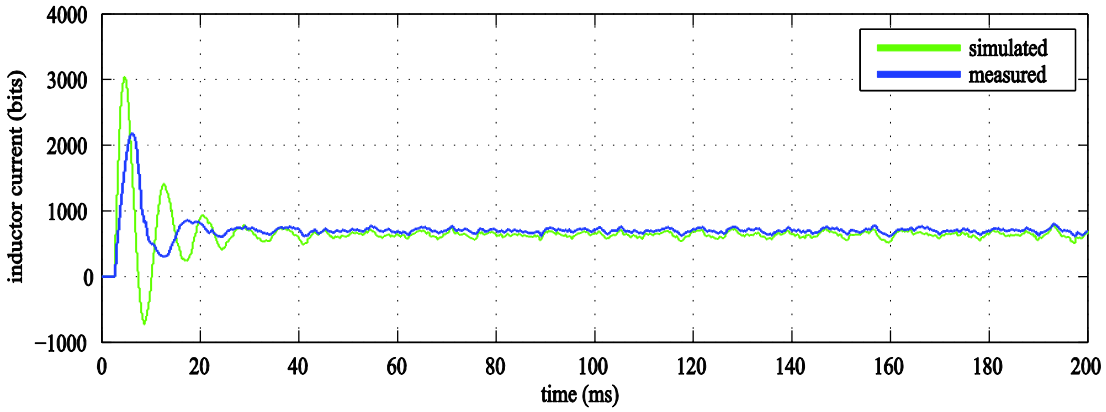


Fig. 46 Modeled and measured step input response comparison

(41). Both showed characteristic second-order LTI system responses but had significant differences with respect to one another. The measured-to-modeled error exceeded 100% during the startup and settled to a 15% error in steady state, as shown in Fig. 47. The continuous time (34), discrete time (36), RLS indentified (40), and FRA plants are all compared in the frequency domain in Fig. 48.

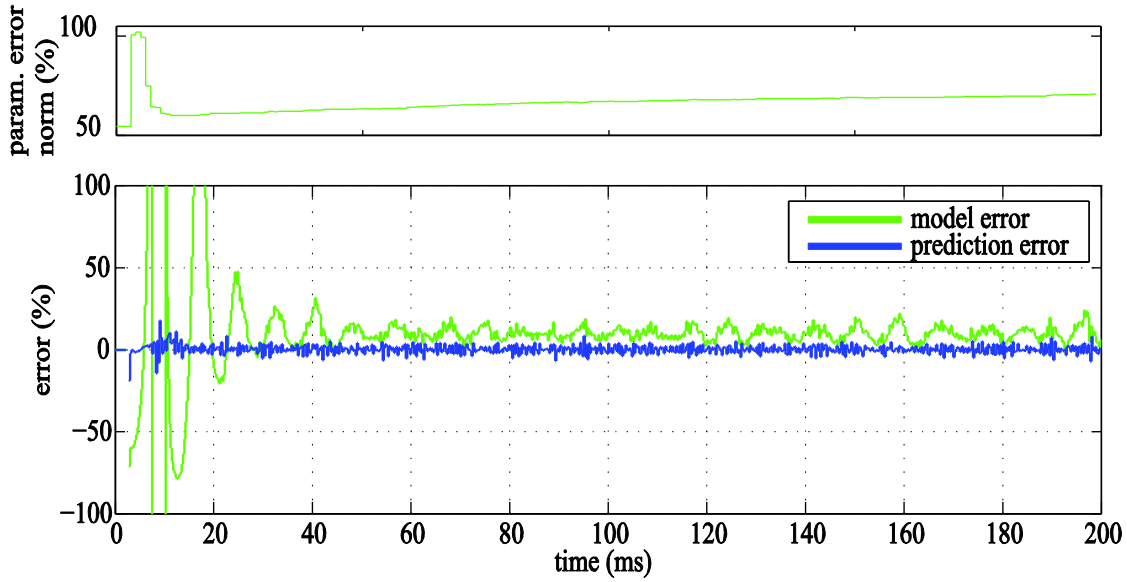


Fig. 47 Hardware system identification performance

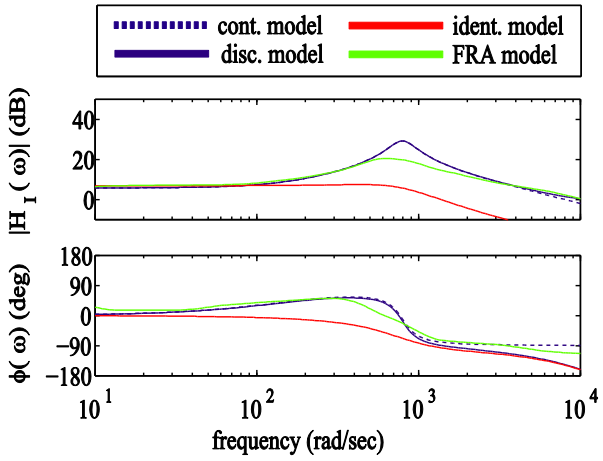


Fig. 48 Model, identified, and frequency sweep comparison for a static 5 Ω load

The second experiment was long term. It cycled through a 200 ms pink-noise sequence and changed the load from 1 to 5 Ω after 2.5 min. The parameter estimate is shown in Fig. 49.The RLS algorithm failed to converge to an accurate model after the 5 Ω load step, a problem that revealed itself in the simulations of Section 3.5.7. The frequency responses for the pre- and postload changes are shown in Fig. 50. Poor system identification performance after abrupt plant parameter changes is a known challenge in many system-identification algorithms [90]. Such algorithms tend to work best for slow parameter variations and large input excitations.

Online system identification, a widely studied academic topic, was implemented on a SMPS. The work had two motivations: first, industry need for fast parameter identification in SMPS, a requirement for high-performance adaptive/optimal control, and second, the lack of hardware experiments in the control literature for abrupt

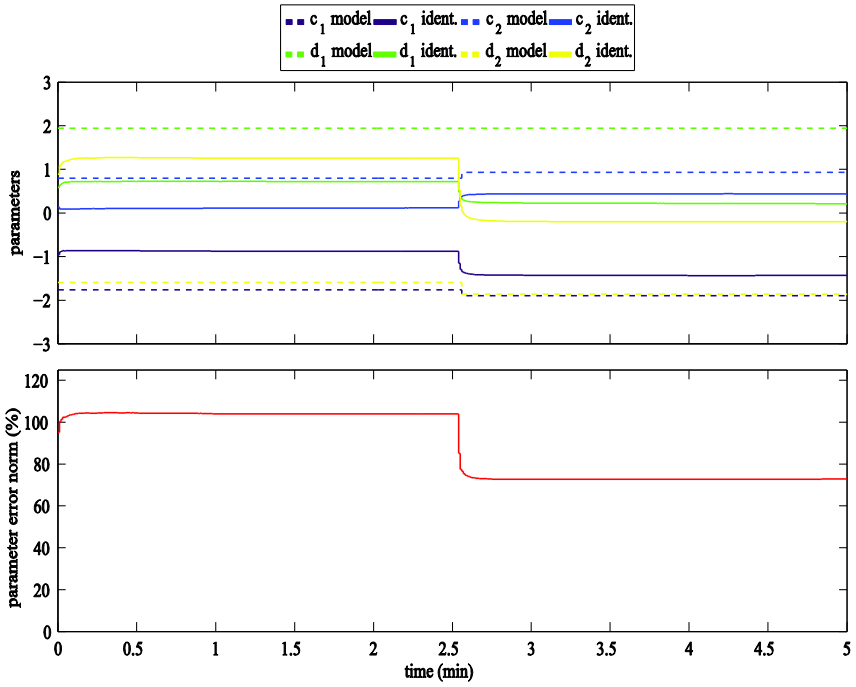


Fig. 49 Identified parameter estimate vector

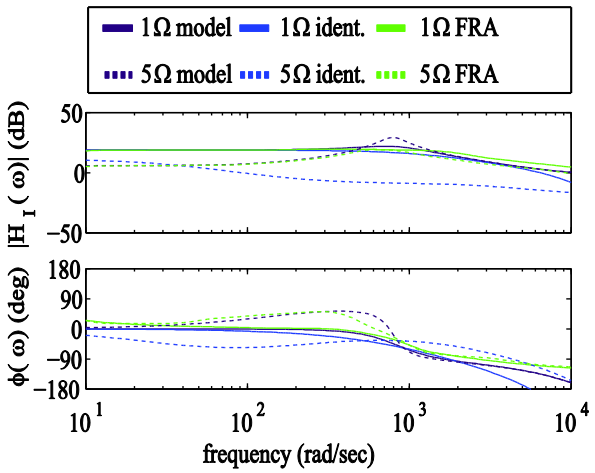


Fig. 50 Model and indentified pre- and postload step frequency response comparisons

parameter changes that show performance metrics, such as parameter error and frequency response. Book chapters and papers on the RLS algorithm tend to emphasize predication error. The algorithm is popular for canceling noise in audio and communications where the metric has most relevance. Sometimes the algorithm is used for tuning poles in the control community where parameter error is more relevant.

This work opened up several potential future research topics that would make online identification more useful for SMPS. One is to devise efficient symbolic mapping of circuit components as they are converted to lumped continuous-time transfer function coefficients and to lumped discrete-time transfer function coefficients. This would allow one to isolate component changes. Second is to apply forgetting factors to tune abrupt load-change identification performance. Third is to use RLS on the output voltage and current to identify the load impedance. Finally, simplified candidate models could be explored. Experimental results showed that the algorithm was effective at finding the dominant pole. A dominant-pole model would reduce needed computation and speed identification.

Section 3.5 investigated the most basic form of the RLS system identification algorithm and presented simulated and hardware results on a synchronous buck converter. Simulations and hardware quickly approximated the plant during startup. The result indicated that RLS would work well in environments where the load stays fixed from startup and changes slowly. For example, RLS could identify a battery charger SMPS whose cell count is fixed throughout charging. However, the RLS method, in its most basic form, did not work well for abrupt parameter change—a common scenario in power electronics. System identification generates the parameter knowledge control needs for performance optimizations; minimum-time is the fastest. Chapter 4 explores performance beyond this limit by actively changing converter parameters and topology.

D.3 Embedded RLS Function

```
function [qHat, yHat,algoClk]=
RlsAlgo(y,u,reset,en,a0,b0,yPhiLength,nParam)
% Recurisive Least Square without Matrix Inversion

%=====INITIALIZE VARIABLES=====
    persistent thetaHat P iAlgo iCB phi yerrI
    if isempty(phi)
        phi=zeros(nParam,2);
    end
    if isempty(iCB)
        iCB=[1 2];
    end
    if isempty(thetaHat)
        thetaHat=zeros(nParam,2);
        thetaHat(:,1)=[-a0;b0];
        thetaHat(:,2)=thetaHat(:,1);
    end
    if isempty(P)
        P=zeros(nParam,nParam,2);
        P(:,:,1)=eye(nParam,nParam);
        P(:,:,2)=eye(nParam,nParam);
    end
    if isempty(iAlgo)
        iAlgo=true;
    end
    if isempty(yerrI)
        yerrI=0;
    end
%=====ALGORITHMS=====
    % run if enable true
    if(en)
        % reset persistent variables
        if (reset)
            phi=zeros(nParam,2);
            thetaHat=zeros(nParam,2);
            thetaHat(:,1)=[-a0;b0];
            thetaHat(:,2)=thetaHat(:,1);
            P=zeros(nParam,nParam,2);
            P(:,:,1)=eye(nParam,nParam);
            P(:,:,2)=eye(nParam,nParam);
            iCB=[1 2];
            iAlgo=~iAlgo;
            % return output
            qHat = thetaHat(:,iCB(1));
            yHat=phi(:,iCB(2))'*thetaHat(:,iCB(1));
            algoClk=iAlgo;
            yerrI=0;
        else
            % shift samples
            phi(2:yPhiLength,iCB(1))=phi(1:yPhiLength-1,iCB(2));
            phi(yPhiLength+2:nParam,iCB(1))=phi(yPhiLength+1:nParam-
1,iCB(2));
            phi(1,iCB(1))=y;
            phi(yPhiLength+1,iCB(1))=u;
            % recursive update
```

```
            yerr=y-phi(:,iCB(2))'*thetaHat(:,iCB(2));
            tempa1=-P(:,:,iCB(2))*phi(:,iCB(1));
            tempa2=phi(:,iCB(1))'*P(:,:,iCB(2));
            tempb1=tempa1*tempa2;
            tempc1=1+tempa2*phi(:,iCB(1));
            P(:,:,iCB(1))=P(:,:,iCB(2))+tempb1/tempc1;
            thetaHat(:,iCB(1))=thetaHat(:,iCB(2))+
P(:,:,iCB(2))*phi(:,iCB(2))*(yerr);
            iAlgo=~iAlgo;
            % return output
            qHat = thetaHat(:,iCB(1));
            yHat=phi(:,iCB(2))'*thetaHat(:,iCB(1));
            algoClk=iAlgo;
            % change circular buffer index
            if iCB(1)==1
                iCB=[2 1];
            else
                iCB=[1 2];
            end
        end
    % if enable false output previous results
    else
        iAlgo=~iAlgo;
        qHat = thetaHat(:,iCB(1));
        yHat=phi(:,iCB(2))'*thetaHat(:,iCB(1));
        algoClk=~iAlgo;
    end
end
```