Adaptive Steady-State Analysis of Circuits Using Wavelets

by
Kris Fedick

A Thesis
Presented to Lakehead University
in Partial Fulfillment of the Requirement for the Degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Thunder Bay, Ontario, Canada

2021-05-02

Abstract

Kristoffer A. Fedick. Adaptive Steady-State Analysis of Circuits Using Wavelets (Under the Supervision of Dr. Carlos Christoffersen).

This thesis presents research into utilizing the sparse representations of waveforms that are possible in the wavelet domain to increase the computational efficiency of the steady-state analysis of electric circuits. The system of non-linear equations that represent the circuits are formulated in the wavelet domain and solved using Newton-Raphson method. Factoring the Jacobian matrix each iteration is a major contributor to the computational time required for solving the circuit equations with Newton-Raphson method. This research aims to reduce the computational time of factoring the Jacobian matrix and has led to the following contributions:

1. A study on the effect of wavelet selection on the sparsity of the Jacobian matrix and nodal variable vectors: Results show that there is no one wavelet that provides the sparsest Jacobian matrices in every case but the Haar wavelet tends to be a good choice if Jacobian matrix sparsity is a concern. However, the time domain provides sparser Jacobian matrices than all of the wavelets tested. Selection of a wavelet to provide the sparsest nodal variable vectors is much more difficult and no one wavelet stood out as providing sparser vectors than the others.

2. A method for increasing the sparsity of the Jacobian matrix via removal of low amplitude entries: The threshold to determine which elements to remove is adaptively controlled during the simulation. Results show that there can be a significant decrease in Jacobian matrix density with adaptive thresholding but the Haar wavelet tends to provide the sparsest matrices with the test cases. The results show that adaptive Jacobian matrix thresholding can lead to a speedup over the non-thresholded wavelet domain steady-state analysis. In some cases, this speedup was enough to lead to a speedup over the time domain when the non-thresholded simulations ran slower than the time domain.

3. Two new methods that reduce the problem size by taking advantage of the sparse representations

that are possible with the nodal variable vectors in the wavelet domain: A unique feature of one of these methods is that it allows for the automatic selection of a wavelet for each nodal variable. Results show a speedup over wavelet domain steady-state analysis for some test cases. There were some test cases where there was a slowdown compared to wavelet domain steady-state analysis which was caused by the computational overhead associated with these methods. With one circuit, the number of columns in the Jacobian matrix was not reduced for most iterations. More work is required to determine if this is due to the method used to select columns from the Jacobian matrix, the method used to control the error introduced into the update vectors by the column reduction method, or if there are some problems that cannot benefit from the column reduction method.

Acknowledgements

I would like to thank all the wonderful people in my life. Without your love and support, this research would not have been possible.

Kris A. Fedick

kafedick@lakeheadu.ca

Contents

List of Figures

List of Tables

List of Abbreviations

ACM model	-	Advanced compact MOSFET model
BSD	-	Berkeley Source Distribution
CANCER	-	Computer Analysis of Nonlinear Circuits, Excluding Radiation
CMOS	-	Complementary Metal-Oxide-Semiconductor
CPU	-	Central Processing Unit
DC	-	Direct Current
DDR3	-	Double Data Rate 3
ECG	-	Electrocardiogram
EKV model	-	Enz Krummenacher Vittoz MOSFET model
EPFL-EKV	-	École Polytechnique Fédérale de Lausanne-Enz Krummenacher Vittoz
HB	-	Harmonic Balance
HP	-	Hewlett-Packard
HRCSSWA	-	Hybrid Reduced Column Steady-State Wavelet Analysis
IC	-	Integrated Circuit
KCL	-	Kirchoff's Current Law
LHS	-	Left Hand Side
LTS	-	Long Term Support
MMIC	-	Monolithic Microwave Integrated Circuit
MOSFET	-	Metal-Oxide-Semiconductor Field-Effect Transistor
Newton method	-	Newton-Raphson Method
PD	-	Partial Discharge
PSSA	-	Periodic Steady-State Analysis
RCSSWA	-	Reduced Column Steady-State Wavelet Analysis
RHS	-	Right Hand Side
RMS	-	Root Mean Squared

S-parameters	-	Scattering Parameters
SDRAM	-	Synchronous Dynamic Random Access Memory
SPICE	-	Simulation Program with Integrated Circuit Emphasis
SSTD	-	Steady-State Time Domain
SSWA	-	Steady-State Wavelet Analysis
WD	-	Wavelet Domain
Y-parameters	-	Admittance Parameters

List of Symbols

$\oslash$ - Used to denote element-by-element division of two vectors which is known as Hadamard division.

$\hat{\mathbf{0}}$ - A vector with NM entries with all entries set to zero.

$\hat{\mathbf{0}}_1$ - A vector with m entries with all entries set to zero.

a - Threshold increase factor.

a_k - Filter coefficient k for $\phi(t)_j$.

$a_{dec,k}$ - Filter coefficient k for $\phi_{dec}(t)_j$.

$a_{rec,k}$ - Filter coefficient k for $\phi_{rec}(t)_j$.

a_{tol} - Absolute tolerance.

A - An $n_r \times n_c$ matrix.

A_{OMP} - The dictionary used to sparsely represent f_{OMP} using the x_{OMP} vector determined with orthogonal matching pursuit.

$A(t)$ - An $N \times N$ time varying matrix that represents an electrical circuit.

α - A constant that is used to control the acceptable amplitude for $\|\hat{\boldsymbol{d}}_\Delta\|_2$.

β - The selected maximum allowable value for $\|\hat{\boldsymbol{d}}_\Delta\|_2$ that can be allowed to consider $\Delta\hat{\boldsymbol{v}}_g$ to be a good estimate of $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$.

b - Threshold decrease factor.

b_k - The filter coefficient k for $\psi(t)_j$.

$b_{dec,k}$ - Filter coefficient k for $\psi_{dec}(t)_j$.

$b_{rec,k}$ - Filter coefficient k for $\psi_{rec}(t)_j$.

$c_{j,k}$ - Scaling coefficient k for the j^{th} scale.

$\boldsymbol{C}$ - $I_M \otimes C$. The Kronecker product of I_M and C.

C - A matrix of nodal capacitances.

C_w - An $M \times M$ matrix that represents the capacitances of a capacitor at each of the M time samples.

$\hat{\boldsymbol{d}}_\Delta$ - An $NM - m$ length vector that represents the error between $\Delta\hat{\boldsymbol{v}}_g$ and $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$.

$\|\hat{\boldsymbol{d}}_\Delta\|_2$ - The wavelet domain residual representing the error between $\Delta\hat{\boldsymbol{v}}_g$ and $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$.

$\|\bar{\boldsymbol{d}}_\Delta\|_2$ - The residual associated with calculating $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$ with HRCSSWA.

d - The right hand side vector of an equation with the form $Ax = d$.

$d_{j,k}$ - Wavelet coefficient k for the j^{th} scale.

d_p - Coefficient used in constructing D_f.

$\boldsymbol{D}$ - $I_N \otimes D$. The Kronecker product of I_N and D.

D - A matrix that performs a derivative operation on a vector.

D_1 - The upper portion of the partitioned D matrix. This is used with the Chebyshev wavelet formulation.

D_2 - The lower portion of the partitioned D matrix. This is used with the Chebyshev wavelet formulation.

D_{B2} - The derivative matrix for the first order backward differences derivative estimate.

D_{B4} - The derivative matrix for the first order backward differences derivative estimate using the two point method.

D_{C2} - The derivative matrix for the first order central differences derivative estimate.

D_{C4} - The derivative matrix for the first order central differences derivative estimate using the four point method.

D_f - A matrix that performs the derivative operation in the frequency domain.

D_{F2} - The derivative matrix for the first order forward differences derivative estimate.

D_{F4} - The derivative matrix for the first order forward differences derivative estimate using the two point method.

D_s - An $NM \times NM$ matrix that represents the derivative operation on nodal variable vector $\bar{v}_s$. This matrix assumes an organization of nodal variables per sample.

Δt - The change in time between time samples.

$\Delta\bar{\boldsymbol{v}}^{j+1}$ - The time domain Newton-Raphson update vector for iteration j.

$\Delta\hat{\boldsymbol{v}}^{j+1}$ - The wavelet domain Newton-Raphson update vector for iteration j.

$\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$ - A vector of all non-zero entries in $\Delta\hat{\boldsymbol{v}}^{j+1}$.

$\Delta\hat{\boldsymbol{v}}_{\sim\Delta}^{j+1}$ - A vector of all zero entries in $\Delta\hat{\boldsymbol{v}}^{j+1}$.

$\Delta\hat{\boldsymbol{v}}_g$ - An m length vector that represents the estimate of $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$.

$\Delta\hat{\boldsymbol{v}}_h^{j+1}$ - Wavelet domain Newton-Raphson update vector that is calculated using $\hat{\boldsymbol{J}}_h$.

$\Delta\hat{\boldsymbol{v}}_{h,i}^{j+1}$ - Element i of $\Delta\hat{\boldsymbol{v}}_h^{j+1}$.

$\Delta\hat{\boldsymbol{v}}_l^{j+1}$ - Element l of $\Delta\hat{\boldsymbol{v}}^{j+1}$.

$\hat{\boldsymbol{\epsilon}}$ - Introduced error vector that results from using $\hat{\boldsymbol{J}}_h$ instead of $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ to calculate the Newton update vector.

f_{OMP} - A vector that is to be decomposed into a sum of dictionary elements. Used in the orthogonal matching pursuit discussion.

$f(a)$ - $f(y)$ evaluated at arbitrary point a.

$f'(a)$ - The derivative of $f(a)$.

$\bar{f}(\bar{v}_k^j)$ - The values of the discretized time domain circuit equation for node k of iteration j.

$f(v_{k,l}^j)$ - The value of the discretized time domain circuit equation sample l of node k.

$\bar{f}_s(\bar{v}_s)$ - A discretized NM element vector representing the nonlinear components of a circuit arranged so that all nodal variables for a given sample are grouped together (nodal variables per sample).

$\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j)$ - The discretized time domain circuit equation for iteration j for steady-state time domain.

$\bar{\boldsymbol{f}}'_{\bar{\boldsymbol{v}}}(\bar{\boldsymbol{v}}^j)$ - The partial derivative of $\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j)$ with respect to each nodal voltage.

$\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$ - The wavelet domain circuit equation for iteration j for steady-state wavelet analysis.

$\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)_k$ - Element k in $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$.

F - Factor matrix found during LU decomposition. This is made up of the L_f and U_f matrix combined.

$\hat{F}(t)$ - A matrix that represents the system of equations with the Chebyshev spline wavelet formulation.

$\hat{F}_1$ - Upper portion of the partitioned Chebyshev spline wavelet formulation.

$\hat{F}_2$ - Lower portion of the partitioned Chebyshev spline wavelet formulation.

$\mathscr{F}\{x(t)\}$ - The Fourier transform of $x(t)$.

$\mathscr{F}^{-1}\{u(f)\}$ - The Inverse Fourier transform of $u(f)$.

$\mathcal{F}$ - A matrix that performs the Fourier Transform on a given sampled time domain signal.

$\mathcal{F}^{-1}$ - A matrix that performs the Inverse Fourier Transform on a given frequency domain signal.

$\boldsymbol{\mathcal{F}}$ - $I_N \otimes \mathcal{F}$. The Kronecker product of I_N and $\mathcal{F}$.

$\boldsymbol{\mathcal{F}}^{-1}$ - $I_N \otimes \mathcal{F}^{-1}$. The Kronecker product of I_N and $\mathcal{F}^{-1}$.

$\hat{\boldsymbol{\delta}}$ - A vector that represents the difference between $\Delta\hat{\boldsymbol{v}}^{j+1}$ and $\Delta\hat{\boldsymbol{v}}_h^{j+1}$

g - A constant used with gyrators.

g_{OMP} - A vector which represents the projection of r_{OMP} onto the columns of A_{OMP}.

$\hat{\boldsymbol{g}}_d^j$ - A vector that, once thresholded, provides an update to the estimate of S_v^{j+1}.

$\hat{\boldsymbol{g}}_v^j$ - A vector that, once thresholded, provides the estimate of S_v^{j+1}.

G - Nodal admittance matrix.

$\boldsymbol{G}$ - $I_M \otimes G$. The Kronecker product of I_M and G.

h - Threshold value. This symbol is used for the adaptive threshold used with the adaptive Jacobian matrix thresholding method as well as for the automatic threshold of RCSSWA and HRCSSWA.

h_s - The threshold used on each block of Ξ and is calculated as: $\gamma \left\| \Xi_{i,j} \right\|_\infty$.

$i(v(t))$ - A vector of voltage dependant nodal currents.

$i_A(v_C(t))$ - Current source A which is dependant on the voltage at node C.

$i_l(t)$ - A vector of representing the time domain nodal current for node l.

$i_l(j\omega)$ - A vector of representing the frequency domain nodal current for node l.

$i_{cap}(t)$ - Capacitor current.

$\overline{\boldsymbol{i}}(\overline{\boldsymbol{v}})$ - A vector of nodal voltage dependant time domain sample currents for each node.

$\overline{\boldsymbol{i}}(\overline{\boldsymbol{v}}^j)$ - A vector of nodal voltage dependant time domain sample currents for each node for iteration j.

i_{in} - Current flowing into a node.

i_{out} - Current flowing out of a node.

$\hat{i}_r$ - Wavelet domain resistor current.

$\hat{i}_{l,o}$ - Wavelet domain initial inductor current.

$\hat{i}_l$ - Wavelet domain inductor current.

$\hat{i}_c$ - Wavelet domain capacitor current.

I - An identity matrix.

I_M - An $M \times M$ identity matrix.

I_N - An $N \times N$ identity matrix.

j_m - The selected scale, or resolution, of a wavelet transform.

$J(W_s \bar{v}_s)$ - Wavelet domain Jacobian matrix used with Newton method when solving for $\bar{v}_s$.

$\overline{\boldsymbol{J}}(\overline{\boldsymbol{v}}^j)$ - The time domain Jacobian matrix at iteration j.

$\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ - The wavelet domain Jacobian matrix at iteration j.

$\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}$ - The entry in $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ located at column l of row k.

$\hat{\boldsymbol{J}}_\Delta$ - A Jacobian matrix made up of columns selected from $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ that are indexed by S_Δ.

$\hat{\boldsymbol{J}}_{\sim\Delta}$ - A Jacobian matrix made up of columns from $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ that are not indexed by S_Δ.

$\hat{\boldsymbol{J}}_h$ - Wavelet domain Jacobian matrix with all elements with amplitudes below h set to 0.

k_m - The maximum number of translations possible at the j^{th} scale of a wavelet transform.

K_i^T - A coefficient row vector of dimension $2^{l+1}+1$ using the Chebyshev polynomial wavelet basis.

K - A matrix of coefficient vectors with each row composed of K_i^T for all $0 < i < N$.

K_v - Re-arranged version of K with $K_v = \left[K_1^T, K_2^T, \ldots, K_N^T\right]^T$.

$L^2(\Re)$ - A space of functions with absolute values that are square integrable.

L - Inductance in Henrys.

L_f - The lower triangular (for $n_r = n_c$) or lower trapezoidal (for $n_r > n_c$) factor matrix found during LU factorization.

L_w - An $M \times M$ matrix that represents the inductances of an inductor at each of the M time samples.

$\boldsymbol{L}$ - An $NM \times m$ lower trapezoidal matrix which results from the LU decomposition of an $NM \times m$ matrix with $m < NM$.

$\boldsymbol{L}_1$ - The upper portion of the partitioned $\boldsymbol{L}$. This is an $m \times m$ lower triangular matrix.

$\boldsymbol{L}_2$ - The lower portion of the partitioned $\boldsymbol{L}$. This is an $(NM - m) \times m$ matrix.

m - The number of non-zero elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$.

m_p - The aliasing integer used with the spectral differentiation method.

M - Number of samples (time domain) or coefficients (wavelet domain) per nodal variable in a given circuit.

n - The number of wavelet coefficients used with the spline wavelets for the Petrov-Galerkin formulation.

n_c - Number of columns.

n_r - Number of rows.

n_{tol} - A selected numerical noise tolerance.

N - Number of nodal variables in a given circuit.

p - A multiplier representing the difference between number of rows and number of columns in matrix A. $p = \frac{n_r}{n_c}$

P - A permutation matrix used to sort the rows of a matrix..

$P_{\mathcal{V}_j} f(t)$ - The approximation of $f(t)$ that using $\phi(t)_j$.

$P_{\mathcal{W}_j} f(t)$ - The approximation of $f(t)$ that using $\psi(t)_j$.

$\boldsymbol{P}_c$ - An $m \times m$ column permutation matrix used with LU decomposition of a tall matrix.

$\boldsymbol{P}_r$ - An $NM \times NM$ row permutation matrix used with LU decomposition of a tall matrix.

$\phi(t)$ - The scaling function, or father wavelet function.

$\phi_{haar}(t)$ - The scaling function of the Haar wavelet.

$\phi(t)_j$ - The j^{th} scaled function of $\phi(t)$.

$\phi_{dec}(t)_{j,k}$ - The deconstruction wavelet scaling function (biorthogonal wavelets).

$\phi(t)_{j,k}$ - The k^{th} translation at the j^{th} scale of $\phi(t)$.

$\phi_{j_0,k,n}$ - The n^{th} sample of $\phi(t)_{j_0,k}$ once it is discretized into M samples.

$\phi_{rec}(t)_{j,k}$ - The reconstruction wavelet scaling function (biorthogonal wavelets).

$\Phi(W_s\bar{v}_s)$ - Wavelet domain circuit function used with Newton method when solving for $\bar{v}_s$.

$\psi(t)$ - The wavelet function, or mother wavelet function.

$\psi_{haar}(t)$ - The wavelet function of the Haar wavelet.

$\psi(t)_{j,k}$ - The j^{th} scaled function of $\psi(t)$.

$\psi(t)_{j,k}$ - The k^{th} translation at the j^{th} scale of $\psi(t)$.

$\psi_{j,k,n}$ - The n^{th} sample of $\psi(t)_{j,k}$ once it is discretized into M samples.

$\Psi_k(t)$ - The l^{th} periodic basis function for the spline wavelet.

$\Psi_{ch}(t)$ - A wavelet basis of level l which has a size of $2^{l+1}+1$.

Ψ_{ch1} - The upper portion of the partitioned Ψ_p matrix. This is used with the Chebyshev wavelet formulation.

Ψ_{ch2} - The lower portion of the partitioned Ψ_p matrix. This is used with the Chebyshev wavelet formulation.

Ψ_{chr} - $[\Psi_{ch}(1) - \Psi_{ch}(-1)]^T$.

$\Psi_{ch}D$ - The transformed derivative matrix. This is used with the Chebyshev wavelet formulation.

Ψ_p - A sorted version of $\Psi_{ch}(t)$ which is given by: $\Psi_p = P\Psi_{ch}(t)$.

$q(v(t))$ - A vector of voltage dependent nodal charges.

$\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}})$ - A vector of nodal voltage dependent time domain sample charges for each node.

$\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}^j)$ - A vector of nodal voltage dependent time domain sample charges for each node for iteration j.

Q_B - An $M \times M$ matrix that performs the integration operation on a vector with M samples.

Q_B^{-1} - An $M \times M$ matrix that performs the differentiation operation on a vector with M samples.

Q_f - A column permutation matrix used to sort the columns in a matrix.

Q_H - Wavelet domain version of Q_B using the Haar wavelet.

Q_H^{-1} - Wavelet domain version of Q_B^{-1} using the Haar wavelet.

r_{OMP} - A residual that is minimized during orthogonal matching pursuit when determining x_{OMP}.

r_{tol} - Relative tolerance.

R_w - An $M \times M$ matrix that represents the resistances of a resistor at each of the M time samples.

R_{F2} - Error that results from using the 2 point forward differences derivative estimate.

R_{F4} - Error that results from using the 4 point forward differences derivative estimate.

R_{B2} - Error that results from using the 2 point backward differences derivative estimate.

R_{B4} - Error that results from using the 4 point backward differences derivative estimate.

R_{C2} - Error that results from using the 2 point central differences derivative estimate.

R_{C4} - Error that results from using the 4 point central differences derivative estimate.

$s(t)$ - A vector of nodal source currents.

$\bar{s}_s$ - A vector of independent sources arranged so all nodal variables for a given sample are grouped together (nodal variables per sample).

$\bar{\boldsymbol{s}}$ - A vector of time domain sample nodal source currents.

$\hat{s}_1$ - A column vector with $\left(2^{l+1}+1\right)N$ entries all equal to zero.

$\hat{s}_2$ - The lower portion of the wavelet domain source vector $s(t)$ using the Chebyshev wavelet formulation.

S_Δ^j - Also known as the support of $\Delta\hat{\boldsymbol{v}}^{j+1}$. This is a list of the positions of all non-zero entries in $\Delta\hat{\boldsymbol{v}}^{j+1}$.

S_v^j - Also known as the support of $\hat{\boldsymbol{v}}^j$. This is a list of the positions of all non-zero entries in $\hat{\boldsymbol{v}}^j$.

S_v^{j+1} - Also known as the support of $\hat{\boldsymbol{v}}^{j+1}$. This is a list of the positions of all non-zero entries in $\hat{\boldsymbol{v}}^{j+1}$.

T - The period of $x(t)$.

T_c - SSWA calculation time.

$T_{\Delta\bar{v}}^j$ - The time taken to calculate $\Delta\bar{\boldsymbol{v}}^{j+1}$ each iteration.

$T_{\Delta\hat{v}}^j$ - The time taken to calculate $\Delta\hat{\boldsymbol{v}}^{j+1}$ each iteration.

T_{hrc} - HRCSSWA calculation time.

T_{HS}^{j} - HRCSSWA overhead time.

T_{rc} - RCSSWA calculation time.

T_{S}^{j} - RCSSWA overhead time.

$\hat{\boldsymbol{t}}$ - The automatically calculated tolerance used with $\hat{\boldsymbol{\epsilon}}$ to adjust threshold h.

$T_{\bar{v}\Delta}^{j}$ - The time taken to calculate $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ each iteration with the reduced column Jacobian.

$T_{\hat{v}\Delta}^{j}$ - The time taken to calculate $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ each iteration with the reduced column Jacobian.

$u(f)$ - The frequency domain representation of $x(t)$.

$\tilde{u}$ - The frequency domain coefficients of $\bar{x}$.

$\tilde{u}_p$ - The p^{th} frequency domain coefficient of $\tilde{u}$.

U_f - The upper triangular factor matrix found during LU factorization.

$\hat{\boldsymbol{u}}$ - A vector with NM elements all set to 1.

$\bar{\boldsymbol{u}}$ - A vector with NM elements all set to 1.

$\boldsymbol{U}$ - An $m \times m$ upper triangular matrix which results from the LU decomposition of an $NM \times m$ matrix with $m < NM$.

$v(t)$ - A vector of nodal voltages.

$\tilde{v}(t)$ - An approximation of $v(t)$ using a spline wavelet.

$v_l(t)$ - The nodal voltage waveform for node l.

$\bar{v}_k$ - The sample voltages of the discretized nodal variable voltage waveforms for node k (time domain).

$\bar{v}_{k,l}$ - The sample voltage for sample l of node k (time domain).

$\bar{v}_s$ - An NM element vector of nodal variables (nodal voltages, branch currents, inductor fluxes, capacitor charges, etc.) arranged so that all nodal variables for a given sample are grouped together (nodal variables per sample).

$\hat{v}_{\phi,k}$ - Scaling coefficient k of $\hat{v}$.

$\hat{v}_{\psi,j,k}$ - Wavelet coefficient k for resolution j of $\hat{v}$.

$\hat{v}_r$ - Wavelet domain resistor current.

$\hat{v}_l$ - Wavelet domain inductor current.

$\hat{v}_{c,o}$ - Wavelet domain initial capacitor voltage.

$\hat{v}_c$ - Wavelet domain capacitor voltage.

$\bar{\boldsymbol{v}}$ - The discretized time domain nodal variable voltage waveforms arranged as sample voltages for each node.

$\bar{\boldsymbol{v}}^j$ - The discretized time domain nodal variable voltage waveforms arranged as sample voltages for each node for iteration j.

$\hat{\boldsymbol{v}}^j$ - The wavelet domain nodal variable vector at iteration j.

$\hat{\boldsymbol{v}}_l^j$ - Element l in $\hat{\boldsymbol{v}}^j$.

$\hat{\boldsymbol{v}}_l^{j+1}$ - Element l in $\hat{\boldsymbol{v}}^{j+1}$.

$\mathcal{V}_j$ - Wavelet domain scaling subspace j.

W - A matrix that performs a wavelet transform on a vector.

W^{-1} - A matrix that performs an inverse wavelet transform on a vector.

W_b - The reconstruction matrix used with biorthogonal wavelets.

W_s - An $NM \times NM$ matrix that performs the wavelet transform on $\bar{v}_s$. This matrix assumes that the vector is arranged so that all nodal variables for a given sample are grouped together (nodal variables per sample).

W_s^{-1} - An $NM \times NM$ matrix that performs the inverse wavelet transform on a vector. This matrix assumes that the vector is arranged so that all nodal variables for a given sample are grouped together (nodal variables per sample).

$\boldsymbol{W}$ - $I_N \otimes W$. The Kronecker product of I_N and W.

$\boldsymbol{W}^{-1}$ - $I_N \otimes W^{-1}$. The Kronecker product of I_N and W^{-1}.

$\boldsymbol{W}_{\Delta}^{-1}$ - A matrix formed using columns from $\boldsymbol{W}^{-1}$ that are indexed by set S_{Δ}^{j+1}.

$\boldsymbol{W}_{\sim\Delta}^{-1}$ - A matrix formed using columns from $\boldsymbol{W}^{-1}$ that are not indexed by set S_{Δ}^{j+1}.

$\boldsymbol{W}_{\Delta 0}^{-1}$ - A matrix copy of $\boldsymbol{W}^{-1}$ with all entries in the columns that are not indexed by S_{Δ}^{j+1} set to zero.

$\boldsymbol{W}_{\sim\Delta 0}^{-1}$ - A matrix copy of $\boldsymbol{W}^{-1}$ with all entries in the columns that are indexed by S_{Δ}^{j+1} set to zero.

$\mathcal{W}_j$ - Wavelet domain wavelet subspace j.

x - Vector of unknowns in an equation of the form $Ax = d$.

x_{OMP} - The sparse representation of f_{OMP} which is determined with orthogonal matching pursuit.

$x(t)$ - Generic function in time (t) used to describe the derivative methods.

$x'(t)$ - Derivative of the generic function in time (t) used to describe the derivative methods.

$x'(t_l)$ - The first order derivative of $x(t)$ with respect to t_l.

$x''(t_l)$ - The second order derivative of $x(t)$ with respect to t_l.

$x'''(t_l)$ - The third order derivative of $x(t)$ with respect to t_l.

$\bar{x}_{df}(t)$ - An interpolating polynomial used with spectral differentiation to interpolate $\bar{x}$ between points.

$\bar{x}'_{df}(t)$ - The derivative of $\bar{x}_{df}(t)$.

$\bar{x}'_{df}$ - Discretized $\bar{x}'_{df}(t)$ with M samples.

$\bar{x}'_{df,n}$ - Sample n of $\bar{x}'_{df}$.

x_l - The value of $x(t)$ at time sample l.

x'_l - The value of $x(t)'$ at time sample l.

$\bar{x}$ - The time samples of $x(t)$ after discretization.

$\bar{x}_p$ - The p^{th} time sample of $\bar{x}$.

Ξ - The wavelet domain convolution matrix for Υ_s which is given as: $\Xi = W_s \Upsilon W_s^{-1}$.

$\Xi_{i,j}$ - The $(i,j)^{\text{th}}$ block of Ξ.

ξ_i - The interpolation points of the splines of the Chebyshev spline wavelets.

y^j - Used in the Appendix describing Newton-Raphson method for the estimate of the root of $f(y)$ at iteration j.

y_f - A vector found with forward substitution as one of the steps to solving for x with LU factorization of a system of equations of the form $Ax = d$.

$\hat{\boldsymbol{y}}$ - Least squares solution to $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1} = -\hat{\boldsymbol{J}}_{\Delta}^{+}\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$ found using LU decomposition.

$\hat{\boldsymbol{y}}_1$ - The upper portion of the partitioned $\hat{\boldsymbol{y}}$. This vector has m entries.

$\hat{\boldsymbol{y}}_2$ - The upper portion of the partitioned $\hat{\boldsymbol{y}}$. This vector has $NM - m$ entries.

$\boldsymbol{Y}$ - A matrix that represents the contributions of linear frequency-defined devices in a circuit.

$\Upsilon(t)$ - A function representing the time domain impulse response of a network of frequency defined components.

$\boldsymbol{\Upsilon}(t)$ - A function representing the time domain impulse response of a network of frequency defined components.

Υ_s - An $NM \times NM$ convolution matrix used for frequency defined components. This matrix operates on $\bar{v}_s$ and assumes all nodal variables for a given sample are grouped together (nodal variables per sample).

z - A set of integers $\{0,1,\cdots,$number of subspaces in $\mathcal{V}_j\}$

$\hat{\boldsymbol{z}}_k$ - The wavelet domain coefficients of the waveform for node k of the vector being thresholded.

Chapter 1

Introduction

For many years, numerical simulation has provided an important tool for the design and testing of circuits. The ability to project the most likely behaviour of a circuit can save much of the time spent in the design stage by allowing designers to not only determine if but also how well their circuits will work before constructing a physical device. Many of the early circuit simulators were proprietary and developed by companies with government contracts. As a result, these simulations were required to have the ability to evaluate the radiation hardness of any given circuit [1]. Then, in 1971, a circuit simulator that utilized a sparse matrix solver was developed. This simulator; called Computer Analysis of Nonlinear Circuits, Excluding Radiation (CANCER)[2]; was capable of simulating circuits that were much larger than previous programs could handle[1, 2]. This program was redeveloped by Nagel during his doctoral studies under Don Pederson and re-named Simulation Program with Integrated Circuit Emphasis (SPICE) [1]. SPICE, unlike its predecessor, is an open source program distributed under the BSD (Berkeley Source Distribution) license [1, 3] and has become widely used and has spawned many more programs, both proprietary (e.g. Multisim and LTSpice) and free to use (e.g. Ngspice distributed under the modified BSD license)[II].

Circuit simulators often are designed to provide multiple types of analysis methods to meet designers needs. The selection of the analysis type is dependent on what characteristics the designer wants to test as well as how quickly and accurately the solution should be found [4, 5]. If the DC operating point for the circuit is desired then a DC analysis could be run or if the response of a circuit to a given input or change in input is desired then a transient analysis is run. Transient analysis is a very accurate simulation method. It can be used to estimate a circuits behavioural characteristics (such as slew rate or cutoff frequency) as well as its response to a given set of inputs. However, transient analysis is inefficient

[II]The BSD license and modified BSD license allow the use of the code with extremely limited restrictions. For more information, see: https://www.freebsd.org/doc/en_US.ISO8859-1/articles/bsdl-gpl/article.html

for the steady-state simulation of circuits that have very long transient responses compared to the period of the input waveforms [4–7].

Circuit simulator performance has been an issue since the first days of simulators. Early computers had limited computational and storage resources which imposed restrictions on both the size and complexity of the circuits that could be simulated. As computer technology advanced, the size and complexity restrictions relaxed, however, these restrictions remained, and likely will remain, important factors when designing circuit simulators. Large and more complex circuits can be simulated by modern computers but, the larger and more complex the circuit, the more time it will take to simulate and the higher the resource drain on the simulating system. Therefore, it is necessary to keep efficiency in mind when designing simulators.

One method of performing steady-state analysis involves dividing the time period of interest into discrete samples, forming a system of circuit equations that allows for all of the samples to be calculated simultaneously, and solving the system of equations using Newton-Raphson method. Each iteration of Newton-Raphson method requires the formation of a Jacobian matrix which represents a system of equations that has to be solved every iteration. The size of the Jacobian matrix increases significantly as the number of elements in the circuit or the number of samples in the time period of interest is increased[I]. If the Jacobian matrix is sparse, taking advantage of sparse matrix storage formats and sparse algorithms can significantly reduce the memory and computational requirements of storage and processing of the Jacobian matrix. This makes Jacobian sparsity an important factor to consider when preparing a circuit simulation.

1.1 Motivation

In recent years, wavelets have been proposed as a possible way to reduce the memory and computational requirements of storage and processing of the Jacobian matrix. This is achieved by taking advantage of the ability of wavelets to sparsely represent a given waveform. These methods work by either reducing the size of the problem (and Jacobian matrix) [8–15] or reducing the number of non-zero elements in the Jacobian matrix [16, 17].

Studies that focused on reducing the size of the Jacobian matrix utilized wavelets that were specifically designed to allow for the circuit equations to be formulated to reduce the number of unknowns to calculate. These methods were successful in reducing the size of the problem but they relied on properties of the wavelets they were designed to use and, as such, cannot be used with other wavelets. Since there are

[I]The Jacobian matrix has $(\#nodes \times \#samples)^2$ elements. Adding elements will not necessarily increase the number of nodes by a large factor but increasing the resolution of the nodal variable waveforms could have a significant effect on the number of samples and, therefore, the number of elements in the Jacobian matrix.

many possible wavelets to choose from, a method that can take advantage of any wavelet basis would be beneficial since it allows for any wavelet basis that exists, or may be developed in the future, to be utilized.

Studies focused on reducing the non-zero elements in the Jacobian matrix operate by setting low amplitude elements from the Jacobian matrix to zero (i.e. thresholding the Jacobian matrix). The threshold is selected before the simulation begins and is performed on the matrix that is used for handling frequency defined components. This method is successful in reducing Jacobian matrix density because the matrix used for handling frequency defined components is a major contributor to the density of the Jacobian matrix. However, this method can lead to an error between the results of the thresholded simulation and a simulation without thresholding if the threshold is too high.

The goal of this research is to develop methods that increase the efficiency of wavelet domain steady-state analysis of electrical circuits with Newton-Raphson method by reducing the number of non-zero elements in the Jacobian matrix and/or reducing the size of the Jacobian matrix which can be used with any wavelet basis.

1.2 Contribution

The purpose of this research is to develop and explore methods for using wavelets to increase the efficiency of calculating the Newton-Raphson update vector when performing steady-state analysis that can be utilized with any wavelet basis. Achieving this purpose resulted in the following developments:

1. A study was performed utilizing many wavelets selected from common wavelet families to determine the effect these wavelets have on the sparisty of the Jacobian matrix and nodal variable waveforms. This study was published in [18] and is further expanded in Chapter 4 by allowing for the sparse matrix pre-ordering determined on the first iteration of the simulation to be re-used every subsequent iteration.

2. A method for thresholding the Jacobian matrix at each iteration of Newton-Raphson method was developed and tested. This method utilizes an automatically controlled threshold that was based on minimizing the error introduced into the Newton-Raphson updates.

3. Methods for reducing the number of columns from the Jacobian matrix that are utilized in calculating the Newton-Raphson updates each iteration were developed and tested. One method utilizes a single selected wavelet basis during simulation while the other adaptively selects wavelets each iteration for each nodal variable waveform.

1.3 Methodology and Technical Challenges

Since there are many potential wavelets to choose from, it is natural to ask: Which wavelet should be used? It is useful to be able to utilize any desired wavelet for an analysis but, when Jacobian matrix and nodal variable vector sparsity is a requirement, it would be counter intuitive to utilize a wavelet that does not provide sparse vectors/Jacobian matrices. Studies to determine the wavelet that provides the sparsest representations of waveforms have been performed for many tasks [19–23]. However, a study comparing many different wavelets to determine the sparsest wavelet for steady-state analysis was not found in the literature. For this reason, a study on wavelets selected from common wavelet families was necessary to determine which wavelet, if any, provides the greatest sparsity Jacobian matrices and nodal waveform vectors. The study, which was published in [18], also explores the effect of thresholding on wavelet domain Jacobian matrix sparsity. This thresholding method was developed to address two issues:

1. How to threshold the Jacobian matrix without introducing error into the final result: This problem was solved by thresholding the Jacobian matrix each iteration of Newton-Raphson method. Any error introduced into the Newton-Raphson update vector is corrected for at a later iteration and there is negligible error introduced into the final result of the simulation.

2. The optimal threshold for one circuit may not be optimum for any other circuit. Additionally, the optimum threshold may change as Newton-Raphson method converges: To address this issue, the threshold is automatically adjusted based on the amount of error introduced into the Newton-Raphson update vector.

The results show the wavelets that provided the highest reduction in Jacobian density with thresholding were wavelets with a higher number of vanishing moments. The Haar wavelet provided the sparsest Jacobian matrices for most cases both with and without thresholding. However, the time domain Jacobian matrices tend to be sparser than the wavelet domain, even when adaptive thresholding is applied. The simulations in this study did not re-use the sparse matrix pre-ordering and this ordering was re-calculated each iteration. To explore the case where the sparse matrix pre-ordering is re-used, the results presented in Chapter 4 were produced by re-using the sparse matrix pre-ordering from the first iteration for the entire simulation.

One of the useful features of wavelets is that they provide sparser representations of waveforms than the time domain. Therefore, a method that takes advantage of the sparsity of the nodal variable vectors was developed and explored. This new method, by design, can be used with any wavelet basis and operates by reducing the columns in the Jacobian matrix used to calculate an estimate of the Newton-Raphson update vectors. This required the development of a method for selecting the columns to be utilized in the

calculation of the update vectors each iteration. Simulation results with the column reduction method show that, although it is possible to achieve a reduction in the number of Jacobian matrix columns each iteration, there is no one wavelet that provides the greatest reduction in Jacobian matrix columns for all cases. Additionally, the optimum wavelet for the column reduction is not easy to select in advance. This made it necessary to develop a method to automatically select an appropriate wavelet. This led to the following developments:

1. The equations were re-formulated in a way that enabled the selection of the Jacobian matrix columns for any selected wavelet without first forming the wavelet domain Jacobian matrix. This makes it possible to select the wavelet that will be expected to provide the sparsest nodal variable vectors without forming multiple wavelet domain Jacobian matrices.

2. Another useful feature of the re-formulated equations is that it is possible to form the Jacobian matrix with far fewer operations. The Jacobian matrix is normally formed in the time domain and then converted to the wavelet domain with pre-multiplication by a forward wavelet transform matrix and post multiplication by an inverse wavelet transform matrix. With the re-formulated equations, the pre-multiplication with the forward wavelet transform matrix is removed and only the selected columns of the inverse wavelet transform matrix are used. The result is a significant reduction in number of operations required to form the wavelet domain Jacobian matrix.

3. Since a wavelet that provides the sparsest nodal variable vector for one node may not necessarily provide the sparsest nodal variable vector for another, a method that can select a wavelet for each node was developed. This is possible because it is not necessary to form the wavelet domain Jacobian matrix until after wavelet selection. To this researchers knowledge, adaptive selection of wavelets for each nodal variable has not been done anywhere in the literature as of the time of the writing of this thesis.

Simulation results show that the column reduction algorithms lead to a slowdown in simulation times over wavelet domain steady-state analysis with the wavelets that led to the fastest simulation times in some cases. However, the simulation times of the method that adaptively selects wavelets for each nodal variable exhibited times that were faster than wavelet domain steady-state analysis simulations with some of the selected test wavelets. Additionally, results with the wavelet domain steady-state analysis simulations show that it is difficult to determine in advance which wavelet will provide the greatest simulation speeds. Thus, the adaptive wavelet selection method can be utilized in cases where the optimum wavelet choice is not known at the cost of a small slowdown in simulation speed compared to wavelet domain steady-state analysis with the optimum choice of wavelet. There is one test circuit where the column reduction

methods do not cause a reduction in columns in some of the tests. This problem, which is one focus of ongoing research, could potentially be overcome by improving the method for selection of columns from the Jacobian matrix and/or improving the method used to control the error introduced into the update vectors when the reduced column Jacobian matrix is used.

The rest of this thesis is organized as follows: Chapter 2 presents a discussion on how the circuit equations are formulated for various components along with a brief description of the time domain steady-state analysis and wavelet domain steady-state analysis methods used in this thesis; Chapter 3 presents a literature review on topics pertaining to the research for this thesis; Chapter 4 presents the results of simulations with the adaptive Jacobian matrix thresholding method and the study on the effect of wavelet selection on Jacobian matrix and nodal variable vector sparsity; Chapter 5 presents the two methods for reducing the average number of columns in the Jacobian matrix along with the results of experimentation with five electrical circuits with some selected wavelets; Chapter 6 concludes this thesis with a summary of the the results of the Jacobian thresholding and column reduction methods along with a discussion of some future avenues for continuing this research. Also included is an Appendix that provides extra details on the derivative estimation methods used in the two studies.

Chapter 2

Background Concepts

This chapter begins with a discussion of the formulation of the circuit equations using nodal analysis and how they are used for time domain periodic steady-state analysis with Newton-Raphson method. Next, a brief introduction to wavelets, how the wavelet transform matrix is included in the circuit equations , and how the time domain periodic steady-state analysis is extended to be performed in the wavelet domain. The chapter ends with a brief introduction to dense and sparse LU factorization and the UMFPack algorithm used in this research for LU factorization. By the end of this chapter, the reader will have an understanding of the periodic steady-state formulation used in this research, what wavelets are and how to include them in the steady-state formulation, and a general idea of how the LU factorization is performed on dense and sparse matrices.

2.1 Nodal Formulation

In order to simulate a circuit, it is necessary to model that circuit mathematically. The circuits in this thesis are modelled using nodal analysis [24–26]. There are many ways that the nodal analysis can be formulated. For example, one could create a vector of all voltages and currents in the circuit and create a matrix, known as a tableau matrix, that represents the nodal equations of the circuit [24]. However, including both voltages and currents in the formulation will lead to a very large matrix and it is often more convenient to select either current or voltages.

The equations in this document are formulated for lumped linear/non-linear and distributed circuit elements using the nodal voltages. This is done by using Kirchoff's current law (KCL), where all the currents entering a node are equal to all the currents leaving the node ($i_{out} = i_{in}$), to form a matrix of nodal admittances for each element connected to each node and a vector of nodal voltages which are used to calculate a vector of currents flowing into each node. For example, consider the circuit in Figure 2.1a. The circuit equation can be written as:

$$Gv(t) = s(t), \tag{2.1}$$

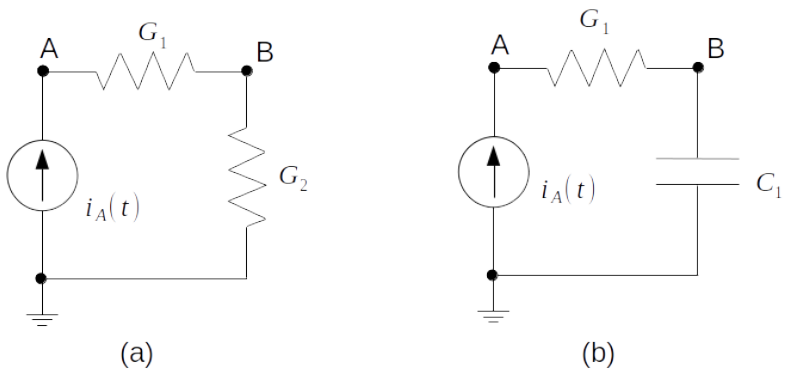

Figure 2.1: Simple resistive network (a) and simple RC network (b)

where G is known as the nodal admittance matrix, $v(t)$ is a vector of nodal voltages, and $s(t)$ is the vector of nodal source currents.

However, circuits are usually much more complex than the simple resistive network of Figure 2.1a. Circuits with capacitors, inductors, voltage sources, and voltage controlled elements such as diodes and transistors are often simulated and increase the complexity of the system of equations.

Capacitors are included by simply applying KCL with the capacitor current equation: $i_{cap}(t) = c\frac{dv(t)}{dt}$, where c is the capacitance in Farads. For example, if we change the resistor connected between node B and ground in Figure 2.1a to a capacitor (Figure 2.1b) the circuit equation becomes:

$$Gv(t) + C\frac{d}{dt}v(t) = s(t), \tag{2.2}$$

where C is a matrix of nodal capacitances.

Many elements, such as inductors and voltage sources can be modelled using special elements called gyrators [24, 27, 28]. A gyrator is made up of two current sources (Figure 2.2a), each related to the voltage at the opposite side of the device with a gyrator constant g which can have any non-negative value. Replacing the voltage source in Figure 2.3a with the ideal voltage source shown in Figure 2.2b converts a voltage source into a current source. The new circuit would look like Figure 2.3b and a new set of terms will be included in the G matrix and $s(t)$ vector. The value of g is selected to keep the gyrator current on a similar order of magnitude as the other branch currents.

An inductor can be built from a gyrator and a capacitor (Figure 2.2c) [24, 27, 28]. This is made possible by relating the inductor voltage ($v_1(t)$ on the left hand side) with the capacitor current ($gv_1(t)$ on the right hand side)

$$i_C = gv_1(t) = C\frac{dv_2(t)}{dt}, \tag{2.3}$$

$$v_L = L\frac{di_L(t)}{dt}. \tag{2.4}$$

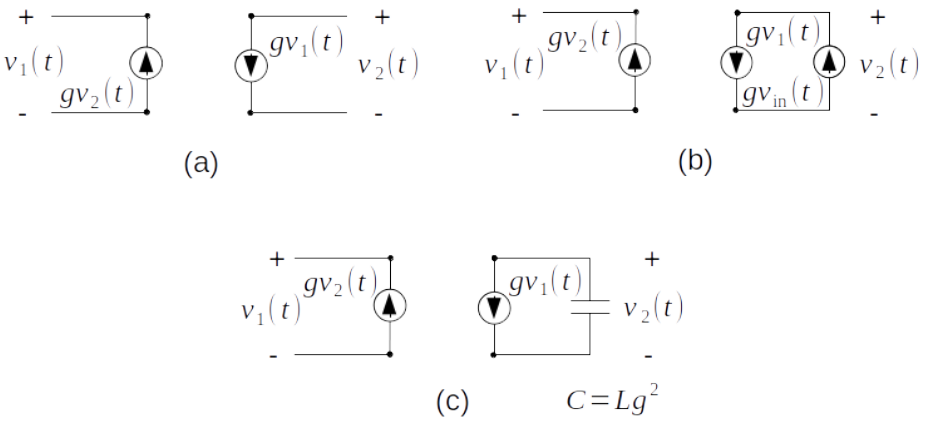

Figure 2.2: Gyrator and gyrator devices. (a) A simple gyrator, (b) an ideal voltage source, and (c) an inductor made from a gyrator and a capacitor. Note: $v_{in}(t)$ is the voltage of the ideal voltage source.

Substituting $v_L = v_1(t)$ and $i_L(t) = gv_2(t)$ into Equation (2.4) results in:

$$\begin{aligned} v_1(t) &= gL\frac{dv_2(t)}{dt}, \\ \frac{dv_2(t)}{dt} &= \frac{v_1(t)}{gL}, \end{aligned} \tag{2.5}$$

where L is the inductance in Henrys of the inductor. Combining Equation (2.3) with Equation (2.5) provides the relation between the inductance of the original inductor and the capacitance of the substituted capacitor in Figure 2.2c:

$$\begin{aligned} gv_1(t) &= C\frac{v_1(t)}{gL}, \\ C &= Lg^2. \end{aligned} \tag{2.6}$$

The value for g that is utilized for the conversion of source voltages into source currents is utilized for the inductor gyrators.

Diodes and transistors can be modelled by combining resistors, capacitors, and current sources. Diodes and transistors also require a voltage dependent set of equations. These equations can be substituted into the models by replacing the elements with functions that describe the characteristics of the device [26]. For example, consider the diode circuit model from Figure 2.4.

The current source $i_1(v_{in}(t))$ and the charge over the capacitor are dependent on the input voltage. The circuit equation is given by:

$$Gv(t) + i(v(t)) + \frac{d}{dt}q(v(t)) = s(t), \tag{2.7}$$

which, when considering circuits with RLC elements as well as diodes and transistors, results in:

$$Gv(t) + C\frac{d}{dt}v(t) + i(v(t)) + \frac{d}{dt}q(v(t)) = s(t). \tag{2.8}$$

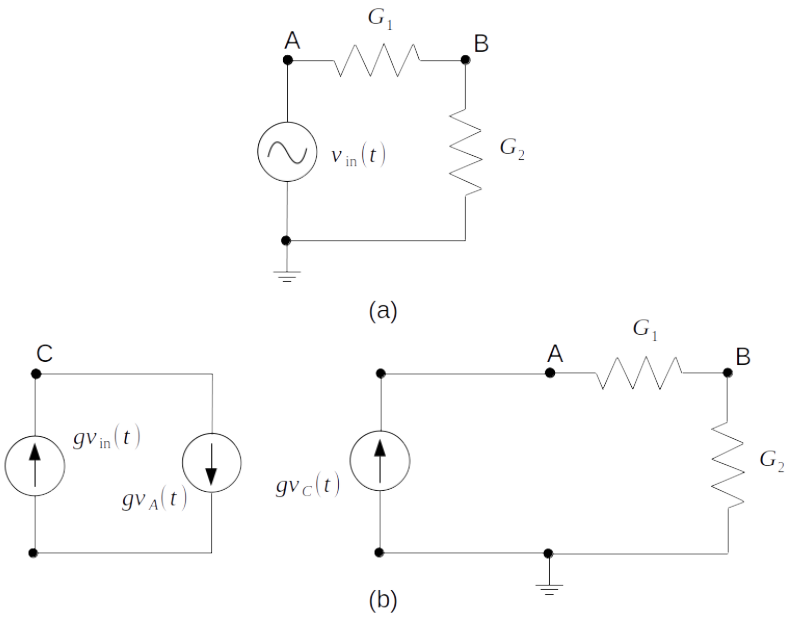

Figure 2.3: Simple resistive circuit with a voltage source. (a) Symbolic representation and (b) Gyrator ideal source.

Where $i(v(t))$ is a vector of voltage dependent nodal currents and $q(v(t))$ is a vector of voltage dependent nodal charges.

Equation (2.8) is sufficient for describing most circuits. However, there are some cases where the characteristics of a device are only known from its measured linear frequency domain response[I]. In this thesis, admittance or Y-parameters are utilized to describe the linear frequency-defined components[II] and are included in the circuit equations with a convolution operation:

$$Gv(t) + C\frac{d}{dt}v(t) + i(v(t)) + \frac{d}{dt}q(v(t)) + \int_{-\infty}^{\infty} \boldsymbol{\Upsilon}(t-\tau)v(\tau)d\tau = s(t), \tag{2.9}$$

where $\int_{-\infty}^{\infty} \boldsymbol{\Upsilon}(t-\tau)v(\tau)d\tau$ is the convolution of the time domain impulse response of the linear frequency defined components, $\boldsymbol{\Upsilon}(t)$, measured as Y-parameters and the time domain voltage vectors for each node. The above approach is a basic introduction to the formulation used by the Cardoon simulator [29]. This design approach was utilized for the simulator so the analysis code only has to deal with three types of elementary devices: voltage-controlled current sources, voltage-controlled charge sources, and independent current sources. Since the Cardoon simulator is used in this thesis to test the proposed wavelet domain steady-state analysis methods, the circuit equations in this thesis are formulated using the same approach. The proposed analysis methods in this thesis operate on the Jacobian matrix after the circuit equations are formulated and, as such, will work with other approaches to formulating the circuit equations.

[I] For example, scattering or S-parameters can be used for transmission lines [25]

[II] Measured S-parameters are converted into Y-parameters.

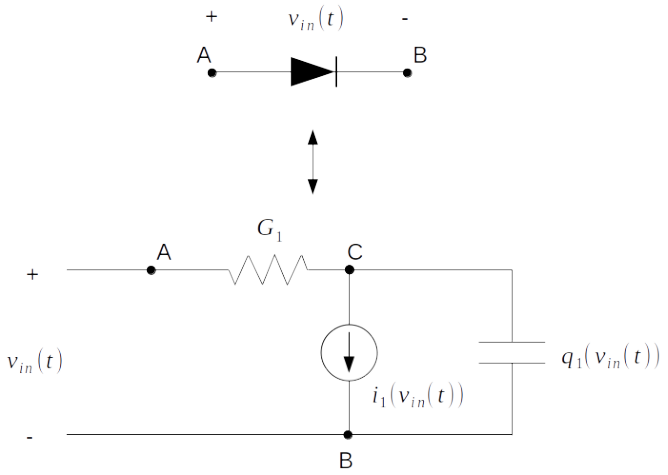

Figure 2.4: Simple Diode circuit model with a voltage controlled current source and a voltage dependent capacitor.

2.2 Periodic Steady-State Time Domain (SSTD)

The periodic steady-state solution is obtained by finding a solution for Equation (2.9) that also satisfies the two-point boundary constraint:

$$v(T) - v(0) = 0. \tag{2.10}$$

In other words, $v(t)$ should be a solution that satisfies Equation (2.9) and the value of $v(t)$ at the end of the time period (time $t = T$) should be the same as the value of $v(t)$ at the beginning of the time period ($t = 0$). The most commonly used method for finding $v(t)$ is the Newton-Raphson method [4], or Newton method, which is used as the solver for this thesis.

The $v(t)$ vector represents the voltages at each node in the circuit which are the variables being calculated by the simulator. When working with time domain steady-state analysis, the nodal voltages are discretized into time samples that each need to be calculated[I]. This requires Equation (2.9) to be modified to account for each of the time samples and can be done by simply including each sample voltage at each node in the voltage vector which leads to two methods of organization: Nodal voltages per sample, and sample voltages per node. For this thesis, sample voltages per node is used and the vectors are structured as shown in Equation (2.11) where N is the number of nodal variables, M is the number of samples for each nodal variable[II], the bar indicates a vector that is in the time domain, a bold symbol indicates a vector of samples for all nodal variables, a single subscript indicates a vector of

[I]For the wavelet domain, the waveform is discretized into its wavelet coefficients.

[II]For this thesis, M and T are known and selected in advance. T is determined by the period of the external sources in this thesis and M is selected as the minimum number of samples required to fully describe the output waveforms with no noticeable sampling error.

samples for a single nodal variable, and the second subscript indicates an individual sample for the nodal variable indicated by the first subscript. A single subscript on a bold symbol (e.g. $\bar{\boldsymbol{v}}_1$), unless otherwise stated, indicates a specific element within the entire vector.

$$v(t) = \begin{bmatrix} v_1(t) \\ v_2(t) \\ \vdots \\ v_N(t) \end{bmatrix} \Longrightarrow \bar{\boldsymbol{v}} = \begin{bmatrix} \bar{v}_1 \\ \bar{v}_2 \\ \vdots \\ \bar{v}_N \end{bmatrix} = \begin{bmatrix} \bar{v}_{1,1} \\ \bar{v}_{1,2} \\ \vdots \\ \bar{v}_{1,M} \\ \bar{v}_{2,1} \\ \bar{v}_{2,2} \\ \vdots \\ \bar{v}_{2,M} \\ \vdots \\ \bar{v}_{N,1} \\ \bar{v}_{N,2} \\ \vdots \\ \bar{v}_{N,M} \end{bmatrix} \tag{2.11}$$

The matrices in Equation (2.9) can be extended to support this organization by using Kronecker products [4] with an identity matrix:

$$(I_M \otimes G)\,\bar{\boldsymbol{v}} + (I_M \otimes C)\,\frac{d}{dt}\bar{\boldsymbol{v}} + \bar{\boldsymbol{i}}(\bar{\boldsymbol{v}}) + \frac{d}{dt}\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}) + [I_N \otimes \mathcal{F}^{-1}] \cdot \boldsymbol{Y} \cdot [I_N \otimes \mathcal{F}]\bar{\boldsymbol{v}} - \bar{\boldsymbol{s}} = \bar{\mathbf{0}}, \tag{2.12}$$

where G is the $N \times N$ nodal admittance matrix, C is the $N \times N$ nodal capacitance matrix, $\mathcal{F}$ and $\mathcal{F}^{-1}$ are $M \times M$ matrices that perform the forward and reverse Fourier transform respectively, $\boldsymbol{Y}$ represents the contributions of linear frequency-defined devices and has a dense[I] $M \times M$ block for each node of the circuit that is connected to a frequency-defined device [17], $\bar{\boldsymbol{i}}(\bar{\boldsymbol{v}})$ is an NM vector of samples of non-linear voltage dependent currents, $\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}})$ is an NM vector of samples of non-linear voltage dependent charges, $\bar{\boldsymbol{s}}$ is an NM vector of independent current sources, I_N is an $N \times N$ identity matrix, I_M is an $M \times M$ identity matrix, and $\bar{\mathbf{0}}$ is an NM vector with all entries set to 0.

Since convolution is equivalent to multiplication in the frequency domain, the voltage vector $\bar{\boldsymbol{v}}$ is converted into the frequency domain so $\boldsymbol{Y}$ can be applied via multiplication and the result is converted back into the time domain. For simplicity, Equation (2.12) can also be written as:

$$\boldsymbol{G}\bar{\boldsymbol{v}} + \boldsymbol{C}\boldsymbol{D}\bar{\boldsymbol{v}} + \bar{\boldsymbol{i}}(\bar{\boldsymbol{v}}) + \boldsymbol{D}\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}) + \boldsymbol{\mathcal{F}}^{-1}\boldsymbol{Y}\boldsymbol{\mathcal{F}}\bar{\boldsymbol{v}} - \bar{\boldsymbol{s}} = \bar{\mathbf{0}}, \tag{2.13}$$

[I] Non-sparse.

where $\boldsymbol{G} = I_M \otimes G$, $\boldsymbol{C} = I_M \otimes C$, $\boldsymbol{\mathcal{F}} = I_N \otimes \mathcal{F}$, $\boldsymbol{\mathcal{F}}^{-1} = I_N \otimes \mathcal{F}^{-1}$, and $\boldsymbol{D} = I_N \otimes D$ where D is a matrix that performs the derivative operation $\frac{d}{dt}$ (see Appendix A).

The number of time samples has a significant effect on the size of the system of equations given in Equation (2.13). The number of time samples depends on the efficiency of the technique used to solve the problem as well as the nature of the signals that are exciting the circuit. For example, if a high frequency and low frequency signal are being simulated, such as the high frequency carrier and low frequency envelope of an amplitude modulated signal, then the time between samples must be very small to accurately describe the high frequency carrier which results in the need for many time samples to show the low frequency envelope [30–34].

The SSTD method is a simple approach in that it solves Equation (2.13) directly in the time domain. Each iteration of Newton method begins with the calculation of :

$$\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j) = \boldsymbol{G}\bar{\boldsymbol{v}}^j + \boldsymbol{C}\boldsymbol{D}\bar{\boldsymbol{v}}^j + \bar{\boldsymbol{i}}(\bar{\boldsymbol{v}}^j) + \boldsymbol{D}\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}^j) + \boldsymbol{\mathcal{F}}^{-1}\boldsymbol{Y}\boldsymbol{\mathcal{F}}\bar{\boldsymbol{v}}^j - \bar{\boldsymbol{s}}, \tag{2.14}$$

where $\bar{\boldsymbol{v}}^j$ is the estimate of $\bar{\boldsymbol{v}}$ at iteration j, $\bar{\boldsymbol{i}}(\bar{\boldsymbol{v}}^j)$ is $\bar{\boldsymbol{i}}(\bar{\boldsymbol{v}})$ evaluated at $\bar{\boldsymbol{v}}^j$, and $\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}^j)$ is $\bar{\boldsymbol{q}}(\bar{\boldsymbol{v}})$ evaluated at $\bar{\boldsymbol{v}}^j$. The Jacobian matrix is then found by taking the derivative of Equation (2.14) with respect to $\bar{\boldsymbol{v}}^j$:

$$\bar{\boldsymbol{f}}'_{\bar{\boldsymbol{v}}}(\bar{\boldsymbol{v}}^j) = \bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j) = \boldsymbol{G} + \boldsymbol{C}\boldsymbol{D} + \boldsymbol{J_I} + \boldsymbol{D}\boldsymbol{J_Q} \tag{2.15}$$

$$\bar{\boldsymbol{J}}_{\boldsymbol{I}} = \begin{bmatrix} \frac{\partial \bar{\boldsymbol{i}}(\bar{\boldsymbol{v}}^j)}{\partial \bar{v}_1^j} & \frac{\partial \bar{\boldsymbol{i}}(\bar{\boldsymbol{v}}^j)}{\partial \bar{v}_2^j} & \cdots & \frac{\partial \bar{\boldsymbol{i}}(\bar{\boldsymbol{v}}^j)}{\partial \bar{v}_N^j} \end{bmatrix} \tag{2.16}$$

$$\bar{\boldsymbol{J}}_{\boldsymbol{Q}} = \begin{bmatrix} \frac{\partial \bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}^j)}{\partial \bar{v}_1^j} & \frac{\partial \bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}^j)}{\partial \bar{v}_2^j} & \cdots & \frac{\partial \bar{\boldsymbol{q}}(\bar{\boldsymbol{v}}^j)}{\partial \bar{v}_N^j} \end{bmatrix}, \tag{2.17}$$

and the Newton update equation for iteration j is:

$$\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\Delta\bar{\boldsymbol{v}}^{j+1} = -\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j), \tag{2.18}$$

2.3 Wavelet Domain (WD)

Wavelets are specially designed functions that are useful for signal and image processing [35–38]. Like sinusoidal functions with the Fourier transform, these functions are used to represent functions in what is known as the wavelet domain. However, unlike sinusoidal functions (Fourier basis), wavelets represent information at different resolutions which are matched to their scale. The result is a tool that allows for information to be broken down, analyzed, and/or processed at different scales or resolutions with a choice of many different basis functions (unlike the Fourier basis which only has one). The ability to represent the signals at different resolutions is known as multiresolution analysis and is an important feature of wavelets.

This section will provide a brief introduction to wavelets. Since the focus of this research is with discrete periodic waveforms, the discussion on the wavelet transform will focus on utilizing wavelets for discrete periodic functions.Multiresolution representations of a function

There are potentially an infinite number of possible basis functions that could be used (including the Fourier basis), however, most of them do not have the special property of being localized in both frequency and time [39]. These basis functions are also designed such that they can be used to form a multiresolution analysis of other functions [35, 40].

Multiresolution analysis is used to represent a given function, $f(t)$, at various levels of resolution [35, 40]. This is achieved by expanding the function in terms of scaling functions,

$$\phi(t)_{j,k} = \frac{1}{\sqrt{2^j}}\phi\left(\frac{t-2^j k}{2^j}\right), \tag{2.19}$$

which are scaled (dilated) by adjusting j and translated by adjusting k to provide approximations of the original function at different levels of detail (i.e. different resolutions). The function, $f(t)$, is approximated in the wavelet domain by orthogonal projections onto different embedded subspaces $\{\mathcal{V}_j\}_{j\in\mathbb{Z}}$[I], each of which are formed by translations of the scaling functions at the same scale, 2^j. Scaling $\phi(t)$ by a factor of 2^j allows the corresponding subspace, $\mathcal{V}_j$, to represent signal features at the resolution 2^{-j}. Thus, scaling up $\phi(t)_{j,k}$ by increasing j will increase the subspace level of $\mathcal{V}_j$ while decreasing the resolution. In the rest of this thesis, scale 2^j is referred to as scale j (or transform level j). Each $\mathcal{V}_j$ has the following properties [35, 40]:

1. There exists a scaling function, $\phi(t)_{0,k}$, such that $\{\phi(t-k)\}_{k\in\mathbb{Z}}$[II] is a Riesz basis of $\mathcal{V}_0$: This allows for the formation of all other subspaces by scaling $\phi(t)_{j,k}$. If $\{\phi(t-k)\}_{k\in\mathbb{Z}}$ forms a Riesz basis of $\mathcal{V}_0$ then $\left\{\frac{1}{\sqrt{2}}\phi\left(\frac{t-2k}{2}\right)\right\}_{k\in\mathbb{Z}}$ forms a Riesz basis of $\mathcal{V}_1$, $\left\{\frac{1}{2}\phi\left(\frac{t-4k}{4}\right)\right\}_{k\in\mathbb{Z}}$ forms a Riesz basis of $\mathcal{V}_2$, and so on. Therefore, we can say that $\left\{\frac{1}{\sqrt{2^j}}\phi\left(\frac{t-2^j k}{2^j}\right)\right\}_{k\in\mathbb{Z}}$ forms a Riesz basis of $\mathcal{V}_j$ with parameter j being referred to as the scale of the subspace $\mathcal{V}_j$. In order to form a Riesz basis, the translations of $\phi(t)_{j,k}$ for all k, have to be linearly independent, therefore:

$$\langle\phi(t)_{j,k}, \phi(t)_{j,l}\rangle = \delta_{k,l} \qquad \forall\,(j,k)\in\mathbb{Z}, \tag{2.20}$$

   where $\delta_{k,l} = \begin{cases} 1 & k=l \\ 0 & otherwise \end{cases}$ and $\langle\phi(t)_{j,k}, \phi(t)_{j,l}\rangle$ is the inner product between $\phi(t)_{j,k}$ and $\phi(t)_{j,l}$[III].

2. $\forall\,(j,k)\in\mathbb{Z}, \qquad f(t)\in\mathcal{V}_j \iff f(t-2^j k)\in\mathcal{V}_j$: This property means that $\mathcal{V}_j$ is invariant by any translation proportional to the scale 2^j.

3. $\forall j\in\mathbb{Z}, \qquad \mathcal{V}_{j+1}\subset\mathcal{V}_j$[IV]: This property means an approximation at a resolution, 2^{-j}, contains all

[I] Note: $\mathbb{Z}$ is a set containing all integers between $-\infty$ and ∞. For the purposes of this discussion, $j\in\mathbb{Z}$ can be taken to mean "for all valid integer values of j".

[II] $\phi(t)_{0,k} = \frac{1}{\sqrt{2^0}}\phi\left(\frac{t-2^0 k}{2^0}\right) = \phi(t-k)$

[III] Note: If $\phi(t)_{j,k}$ is real (i.e. contains no complex numbers), then $\langle\phi(t)_{j,k}, \phi(t)_{j,l}\rangle = \phi(t)_{j,k}\cdot\phi(t)_{j,l}$. The scaling functions of the wavelets used in this thesis contain only real numbers.

[IV] $\mathcal{V}_{j+1}\subset\mathcal{V}_j$ means that $\mathcal{V}_{j+1}$ is a subset of $\mathcal{V}_j$.

necessary information required to compute an approximation at a coarser resolution $2^{-(j+1)}$. Thus, we can express any function in $\mathcal{V}_j$ in terms of basis functions in $\mathcal{V}_{j-1}$ and [35]:

$$\phi(t)_j = \sum_{k=-\infty}^{\infty} a_k \phi(t)_{j-1,k}, \tag{2.21}$$

where $a_k \in \mathbb{Z}$ are known as the filter coefficients and are a square summable sequence.

4. $f(t) \in \mathcal{V}_j \iff f(\frac{t}{2}) \in \mathcal{V}_{j+1}$: Dilating a function in $\mathcal{V}_j$ by a factor of 2 will enlarge the details by 2 and $\mathcal{V}_{j+1}$ will define an approximation at a courser resolution.

5. $\lim_{j\to\infty} \mathcal{V}_j = \bigcap_{j=-\infty}^{\infty} \mathcal{V}_j = \{0\}$[I]: This property indicates that all details of f are lost as the resolution, 2^{-j}, approaches zero.

6. $\cup_{j\in\mathbb{Z}} \mathcal{V}_j$[II] is dense in $L^2(\Re)$: This property forces the signal approximation to converge to the original signal as the resolution approaches ∞.

If the above properties are met for $\phi(t)_{j,k}$, then an approximation of a function, f(t), for any scale j can be obtained by projecting $f(t)$ over $\phi(t)_{j,k}$ [35, 40]:

$$P_{\mathcal{V}_j} f(t) = \sum_{k=-\infty}^{\infty} c_{j,k} \phi(t)_{j,k}, \tag{2.22}$$

$$c_{j,k} = \langle f(t), \phi(t)_{j,k} \rangle, \tag{2.23}$$

where the inner products, $c_{j,k}$, provide a discrete approximation at scale j and each of the k elements in $c_{j,k}$ are known as scaling coefficients for scaling subspace j or just scaling coefficients.

An interesting property of the scaling coefficients is that, as the scale is increased, the number of coefficients required to approximate the signal is decreased. This property is easily illustrated with an example. The Haar wavelet is the simplest and oldest wavelet [37]. The scaling function is a simple box function:

$$\phi_{haar}(t)_0 = \begin{cases} 1 & 0 < t < 1 \\ 0 & otherwise. \end{cases} \tag{2.24}$$

Figure 2.5 shows the scaling coefficients for an example function which is approximated with the Haar wavelet with three different scales. With each increase in scale, the number of coefficients describing the

[I] $\bigcap_{j=-\infty}^{\infty} \mathcal{V}_j$ indicates the intersection of subspaces that are nested within $\mathcal{V}_j$. As the scale increases, the number of higher scale subspaces nested within $\mathcal{V}_j$ decreases. The intersection of subspaces indicates where details are retained as the scale is increased. As the set that represents the intersection between subspaces decreases, the number of details that are retained decreases as well. If an increase in scale causes the intersection to become $\{0\}$ then there are no details retained with the new scale.

[II] i.e. The union of all subspaces $\mathcal{V}_j$.

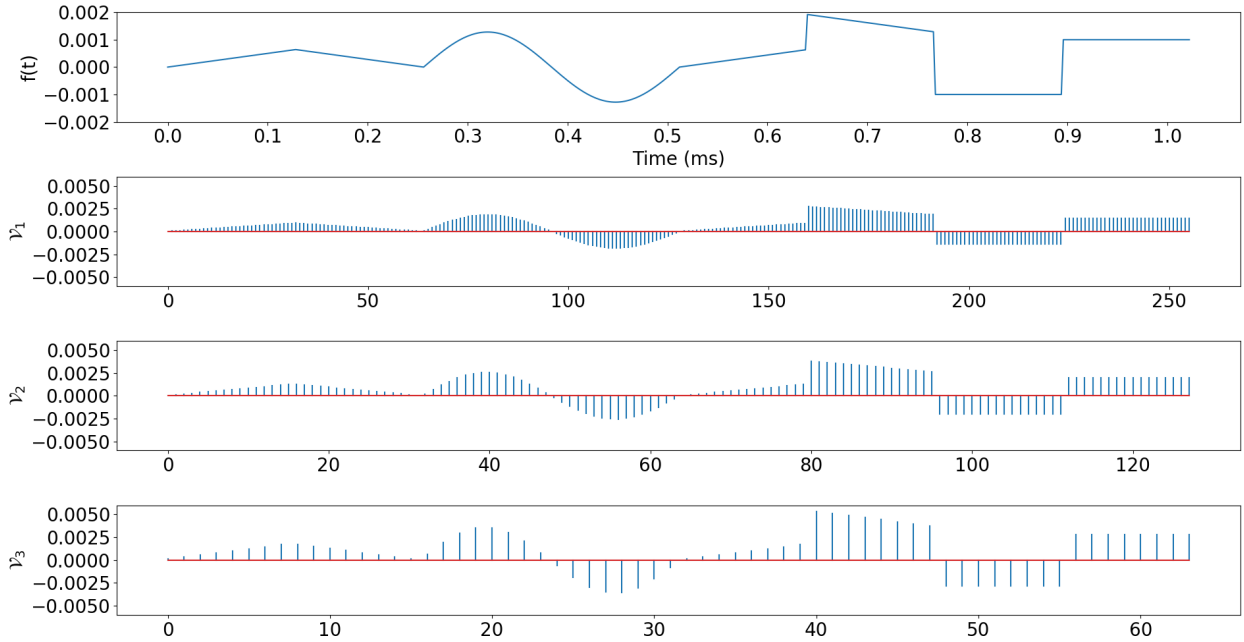

Figure 2.5: Scaling coefficients of a function, $f(t)$, with the Haar wavelet at three scales. The resolution becomes coarser with each increase in scale. The wavelet transform for this example was performed with the discrete wavelet transform described in Section 2.3.4 with an $f(t)$ discretized into 512 samples.

signal is divided by 2. However, this sparsity comes at the cost of a loss of details in the time domain representation of the signal, shown in Figure 2.6.

2.3.1 Orthogonal wavelets

Adjusting the scale of the transform is useful for representing a function at different resolutions and allowing for a sparser representation of a function. However, what if we are interested in the details between scaling subspaces? Indeed, the wavelet transform would not be as useful if it was impossible to recover the original function. If a new subspace, $\mathcal{W}_j$, that represents the details that are lost between scaling subspaces is introduced, it is possible to express $\mathcal{V}_j$ as a combination of the scaling subspace at the next higher scale (lower resolution), $\mathcal{V}_{j+1}$, and the wavelet subspace at the next higher scale (lower resolution), $\mathcal{W}_{j+1}$ [35, 40]:

$$\mathcal{V}_j = \mathcal{V}_{j+1} \oplus \mathcal{W}_{j+1}; \qquad \mathcal{V}_{j+1} \perp \mathcal{W}_{j+1}, \tag{2.25}$$

To describe $\mathcal{W}_j$, we introduce another function, $\psi(t)_{j,k}$, known as the wavelet function:

$$\psi(t)_{j,k} = \frac{1}{\sqrt{2^j}} \psi\left(\frac{t - 2^j k}{2^j}\right). \tag{2.26}$$

This function has the same 6 properties as $\phi(t)$, discussed in Section 2.3, which leads to:

$$P_{\mathcal{W}_j} f(t) = \sum_{k=-\infty}^{\infty} d_{j,k} \psi(t)_{j,k}, \tag{2.27}$$

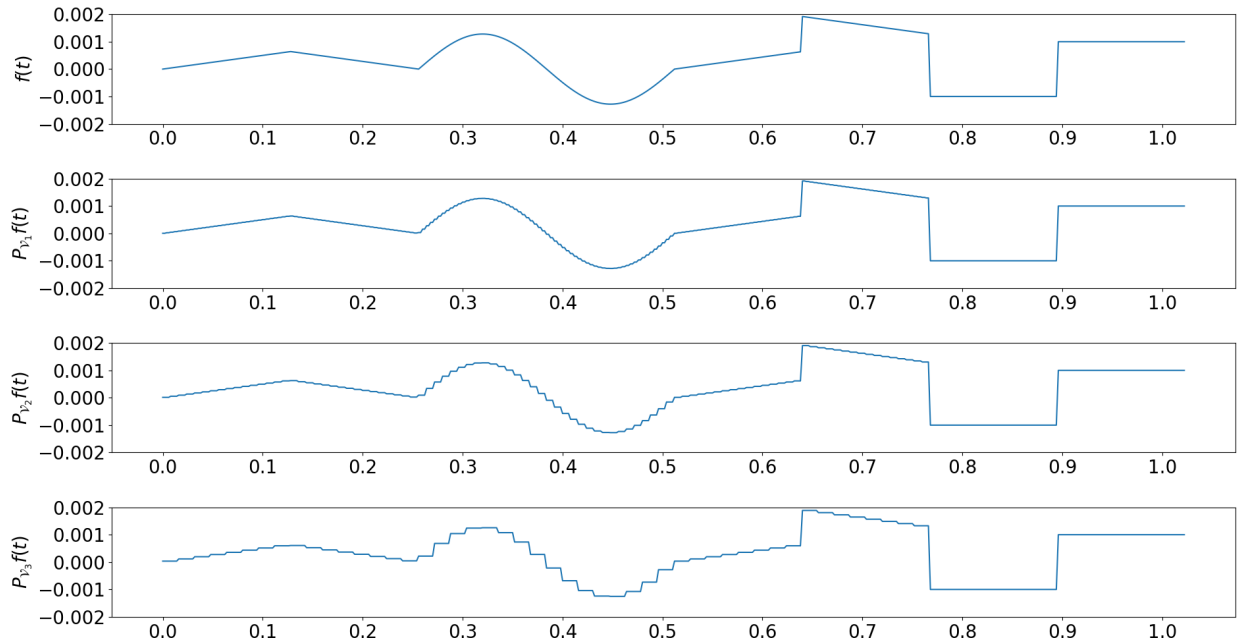

Figure 2.6: Approximation of a function, $f(t)$, with the Haar wavelet at three scales. The resolution becomes coarser with each increase in scale. The wavelet transform for this example was performed with the discrete wavelet transform described in Section 2.3.4 with an $f(t)$ discretized into 512 samples.

$$d_{j,k} = \langle f(t), \psi(t)_{j,k} \rangle \,, \tag{2.28}$$

where the inner products, $d_{j,k}$, provide the details at scale 2^j with each of the k coefficients in $d_{j,k}$ being known as the wavelet coefficients of subspace j. Combining $P_{\mathcal{V}_j} f(t)$ and $P_{\mathcal{W}_j} f(t)$ at one scale (resolution) will result in an an approximation of the function at the next lower scale (higher resolution):

$$P_{\mathcal{V}_j} f(t) = P_{\mathcal{V}_{j+1}} f(t) + P_{\mathcal{W}_{j+1}} f(t). \tag{2.29}$$

Combining $P_{\mathcal{V}_{j+1}} f(t)$ and $P_{\mathcal{W}_{j+1}} f(t)$ in this way means that any details that are lost by the approximation of $f(t)$ with $P_{\mathcal{V}_{j+1}} f(t)$, instead of $P_{\mathcal{V}_j} f(t)$, are preserved in $P_{\mathcal{W}_{j+1}} f(t)$. This process can be repeated until any scale, k, has been reached:

$$\begin{aligned} P_{\mathcal{V}_j} f(t) &= P_{\mathcal{V}_{j+1}} f(t) + P_{\mathcal{W}_{j+1}} f(t) \\ &= P_{\mathcal{V}_{j+2}} f(t) + P_{\mathcal{W}_{j+2}} f(t) + P_{\mathcal{W}_{j+1}} f(t) \\ &= P_{\mathcal{V}_{j+3}} f(t) + P_{\mathcal{W}_{j+3}} f(t) + P_{\mathcal{W}_{j+2}} f(t) + P_{\mathcal{W}_{j+1}} f(t) \\ &= P_{\mathcal{V}_k} f(t) + \sum_{l<k} P_{\mathcal{W}_l} f(t), \end{aligned} \tag{2.30}$$

From property 6 in Section 2.3, we know that as the scale approaches $-\infty$, $P_{\mathcal{V}_j} f(t) \to f(t)$ and $f(t)$ can be expressed as a combination of $P_{\mathcal{V}_j} f(t)$ at any choice of j and all wavelet subspaces at and below j:

$$f(t) = P_{\mathcal{V}_j} f(t) + \sum_{l \le j} P_{\mathcal{W}_l} f(t). \tag{2.31}$$

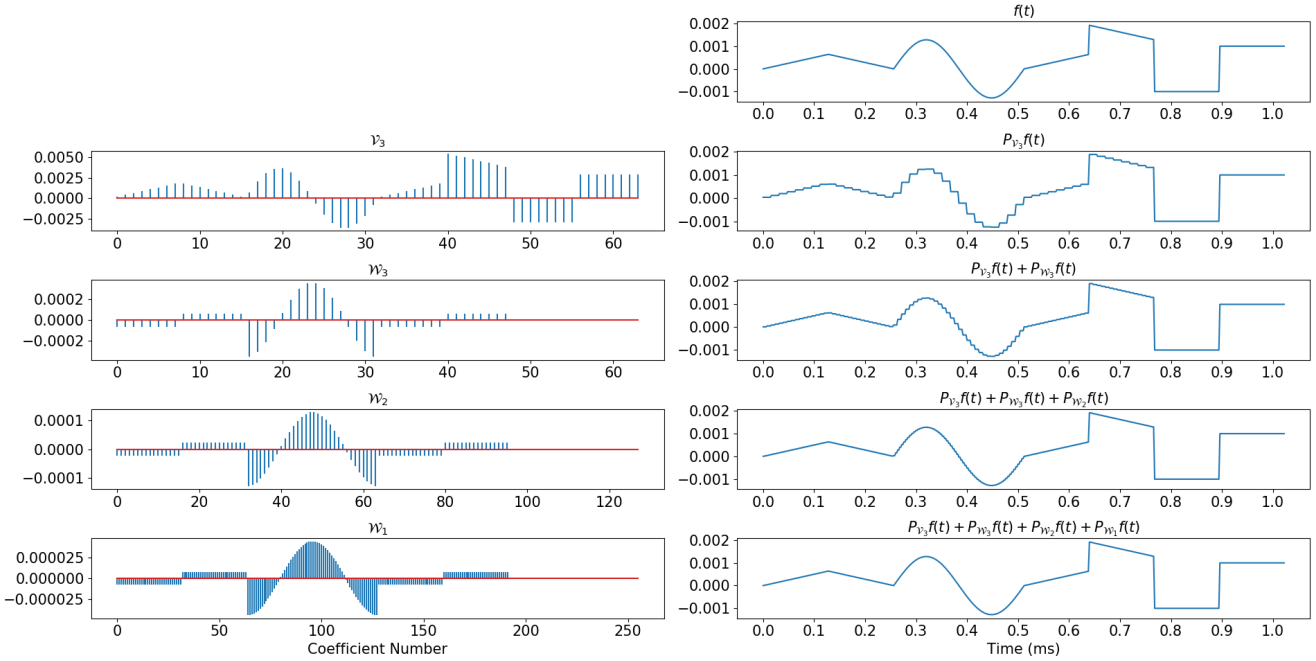

Figure 2.7: Approximation of a function, $f(t)$, with the Haar wavelet with both the scaling and wavelet functions at scale $j = 3$. On the left are the scaling and wavelet coefficients for $\mathcal{V}_3$, $\mathcal{W}_3$, $\mathcal{W}_2$, and $\mathcal{W}_1$. On the right are the time domain estimates with $\mathcal{V}_3$ as the scaling subspace and the effect of the inclusion of each wavelet subspace. The wavelet transform for this example was performed with the discrete wavelet transform described in Section 2.3.4 with an $f(t)$ discretized into 512 samples.

This leads to:

$$f(t) = \sum_{k=-\infty}^{\infty} \langle f(t), \phi(t)_{j,k} \rangle \, \phi(t)_{j,k} + \sum_{l \leq j} \sum_{k=-\infty}^{\infty} \langle f(t), \psi(t)_{l,k} \rangle \, \psi(t)_{l,k}. \tag{2.32}$$

Figure 2.7 shows an example of the signal from Figure 2.6 with the wavelet subspaces included in the approximation. The scaling subspace, $\mathcal{V}_3$, provides a rough approximation that is improved with the addition of each higher resolution wavelet subspace.

Determination of $\psi(t)$ is achieved by taking advantage of the properties of $\mathcal{V}_j$ and $\mathcal{W}_j$. Since $\mathcal{W}_j$ is contained in $\mathcal{V}_{j-1}$, $\mathcal{W}_j$ describes the higher resolution details that are missing from $\mathcal{V}_j$ that are required to form $\mathcal{V}_{j-1}$. This means that the wavelet function at one resolution can be expressed in terms of the scaling function at the next higher resolution [35, 40]:

$$\psi(t)_j = \sum_{k=-\infty}^{\infty} b_k \phi(t)_{j-1,k}, \tag{2.33}$$

where b_k is a set of filter coefficients for the wavelet function. The wavelet function is, by design, orthogonal

to the scaling function[I] and can be expressed as [35]:

$$\psi(t)_j = \sum_{k=0}^{N_a-1} (-1)^k a_{N_a-1-k}\phi(t)_{j-1,k}, \tag{2.34}$$

where N_a represents the number of filter coefficients and $b_k = (-1)^k a_{N_a-1-k}$.

2.3.2 Construction of wavelet systems

Constructing the wavelet system is achieved by selecting a function, $\phi(t)$, and calculating values for a_k and b_k in Equations (2.21) and (2.33). This process is best illustrated with an example. Consider the Haar wavelet which has the scaling function described in Equation (2.24). From property 3 described in Section 2.3, $\phi_{haar}(t)$ at one resolution can be described by the combination of all translations of $\phi_{haar}(t)$ at the next higher resolution. The filter coefficients for the scaling function are calculated by first dilating $\phi_{haar}(t)_0$ and finding its translations[II]:

$$\phi_{haar}(t)_{-1,0} = \begin{cases} \sqrt{2} & 0 < t < 0.5 \\ 0 & otherwise, \end{cases} \tag{2.35}$$

$$\phi_{haar}(t)_{-1,1} = \begin{cases} \sqrt{2} & 0.5 < t < 1 \\ 0 & otherwise. \end{cases} \tag{2.36}$$

From Equation (2.21), $\phi_{haar}(t)_0 = \sum_{k=0}^{1} a_k \phi_{haar}(t)_{0-1,k} = a_0\phi_{haar}(t)_{-1,0} + a_1\phi_{haar}(t)_{-1,1}$ and:

$$\phi_{haar}(t)_0 = \begin{cases} a_0\sqrt{2} & 0 < t < 0.5 \\ a_1\sqrt{2} & 0.5 < t < 1 \\ 0 & otherwise. \end{cases} \tag{2.37}$$

Substitution of $a_0 = a_1 = \frac{1}{\sqrt{2}}$ into Equation (2.37) results in Equation (2.24).

The wavelet function, $\psi_{haar}(t)$, is found by substituting $j = 0$ and $\phi_{haar}(t)_0$ into Equation (2.34):

$$\psi_{haar}(t) = \sum_{k=0}^{1} (-1)^k a_{N_a-1-k}\phi_{haar}(t)_{0-1,k} = a_1\phi_{haar}(t)_{-1,0} - a_0\phi_{haar}(t)_{-1,1}. \tag{2.38}$$

Substituting $a_0 = a_1 = \frac{1}{\sqrt{2}}$, Equation (2.35), and Equation (2.36) into Equation (2.38) results in the wavelet function for the Haar wavelet:

$$\psi_{haar}(t) = \begin{cases} 1 & 0 < t < 0.5 \\ -1 & 0.5 < t < 1 \\ 0 & otherwise. \end{cases} \tag{2.39}$$

[I] From the requirement in Equation (2.25): $\mathcal{V}_{j+1} \perp \mathcal{W}_{j+1}$

[II] By setting $j = -1$ in Equation (2.19).

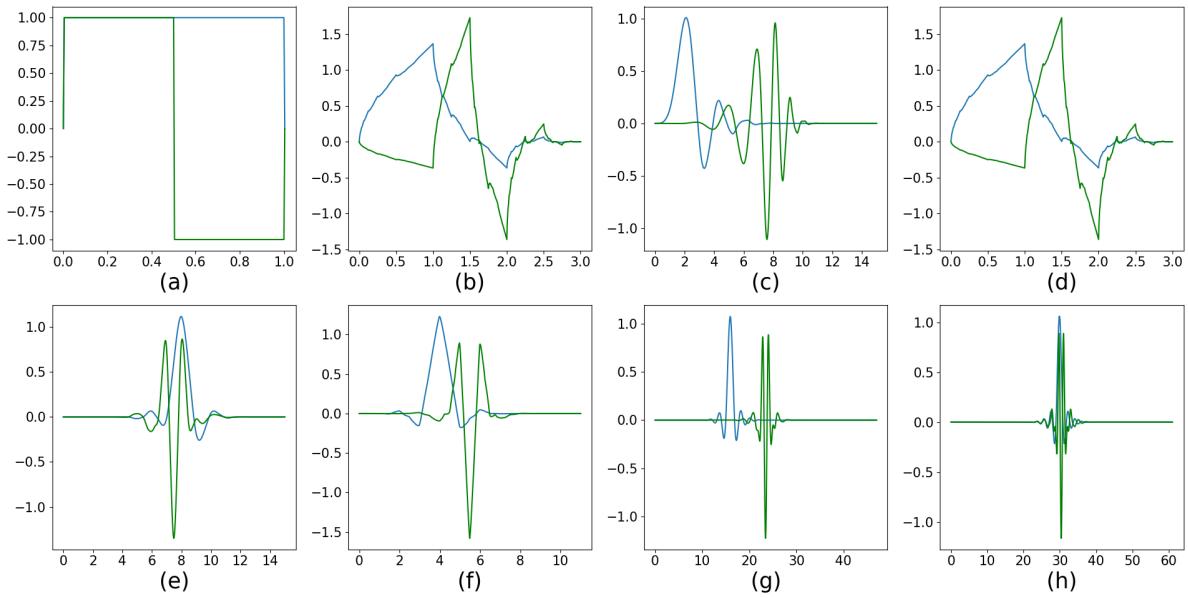

Figure 2.8: Commonly used orthogonal wavelets. Scaling function in blue and wavelet functions in green. (a) Haar, (b) 2 vanishing moment Daubechies, (c) 8 vanishing moment Daubechies, (d) 2 vanishing moment Symlet, (e) 8 vanishing moment Symlet, (f) 2 vanishing moment Coiflet, (g) 8 vanishing moment Coiflet, and (h) discrete Meyer.

The filter coefficients are: $a_0 = a_1 = \frac{1}{\sqrt{2}}$, $b_0 = \frac{1}{\sqrt{2}}$, $b_1 = -\frac{1}{\sqrt{2}}$. The scaling and wavelet functions for the Haar wavelet, along with selected wavelets from other orthogonal families, are shown in Figure 2.8.

Many families of wavelets have been developed. These families use specially designed $\phi(t)$ functions and/or impose special properties on $\phi(t)$. Some of the common families, which are used in this thesis, are the Daubechies, Coiflets, Symlets, and discrete Mayer. The following paragraphs contain a short description of these wavelet families. For more information on the design of these wavelets, along with their filter coefficients, see [40].

The Daubechies wavelet families are designed to have a compact support (i.e. a limited number of non-zero elements exist in $\phi(t)$) for any given number of vanishing moments. A wavelet function $\psi(t)$ has p vanishing moments if [40]:

$$\int_{-\infty}^{\infty} t^k \psi(t) dt = 0 \qquad \text{for } 0 \leq k < p, \tag{2.40}$$

which means that $\psi(t)$ is orthogonal to any polynomial of degree $p-1$. This means that a wavelet with p vanishing moments will be able to reproduce a polynomial of degree $p-1$ using only scaling functions. Indeed, the filter coefficients of the Daubechies wavelets can be derived by imposing the requirement that a polynomial of degree $p-1$ can be represented by translations of the scaling function [35]. Two examples of this wavelet are shown in Figure 2.8b and c.

The Symlet families are similar to the Daubechies wavelet (they are a modified version of the Daubechies

wavelet) but have been designed to be more symmetrical than the Daubechies wavelets [20, 38]. Two examples of this wavelet are shown in Figure 2.8d and e. The Coiflet family, like the Daubechies and Symlet families, is designed to provide p vanishing moments and a minimum support. However, the support of the Coiflet family tends to be larger than the Daubechies and Symlet families and the Coiflet wavelet also requires [38, 40]:

$$\int_{-\infty}^{\infty} \phi(t)dt = 1. \tag{2.41}$$

Two examples of the Coiflet wavelet are shown in Figure 2.8f and g.

The Dmey family (Discrete Meyer family) is a discrete approximation to the Meyer wavelet used with the discrete wavelet transform. This approximation is necessary because the Meyer wavelet is not compactly supported [20]. The Meyer wavelet was designed to be frequency band-limited and have a Fourier transform that is smooth. The Meyer wavelet has infinite support (it does not decay in time) while its Fourier transform has compact support. For this reason, its transform is normally performed in the Fourier domain. It is also the reason that it is necessary to use a discrete approximation when utilizing the discrete wavelet transform. An example of this wavelet is shown in Figure 2.8h.

2.3.3 Biorthogonal wavelets

Biorthogonal wavelets are a family of wavelets that is comprised of two sets of wavelet functions. One set, called the deconstruction functions, are used to transform a time domain function into the wavelet domain while the other set, called the reconstruction functions, are used to rebuild the time domain function from its wavelet domain representation. This means that the scaling and wavelet function sets are designed to be orthogonal to each other [40, 41]:

$$\langle \phi_{dec}(t)_{j,k}, \phi_{rec}(t)_{j,l} \rangle = \delta_{k,l} \qquad \forall\, (j,k) \in \mathbb{Z}, \tag{2.42}$$

$$\langle \psi_{dec}(t)_{j,k}, \psi_{rec}(t)_{j,l} \rangle = \delta_{k,l} \qquad \forall\, (j,k) \in \mathbb{Z}. \tag{2.43}$$

The same condition in Equation (2.21) applies to the scaling function sets:

$$\phi_{dec}(t)_j = \sum_{k=-\infty}^{\infty} a_{dec,k} \phi_{dec}(t)_{j+1,k}, \qquad \phi_{rec}(t)_j = \sum_{k=-\infty}^{\infty} a_{rec,k} \phi_{rec}(t)_{j+1,k}, \tag{2.44}$$

where $a_{dec,k}$ and $a_{rec,k}$ are the k^{th} deconstruction and reconstruction filter coefficients, respectively. However, the deconstruction wavelet functions depend on the filter coefficients of the reconstruction scaling functions and vice versa [41].

$$\psi_{dec}(t)_j = \sum_{k=0}^{N_a-1} (-1)^k a_{rec,N_a-1-k} \phi_{dec}(t)_{j+1,k}, \qquad \psi_{rec}(t)_j = \sum_{k=0}^{N_a-1} (-1)^k a_{dec,N_a-1-k} \phi_{rec}(t)_{j+1,k}. \tag{2.45}$$

A function, $f(t)$, is expressed in the wavelet domain with biorthogonal wavelets as:

$$f(t) = \sum_{k=-\infty}^{\infty} \langle f(t), \phi_{dec}(t)_{j,k} \rangle \, \phi_{rec}(t)_{j,k} + \sum_{l \le j} \sum_{k=-\infty}^{\infty} \langle f(t), \psi_{dec}(t)_{l,k} \rangle \, \psi_{rec}(t)_{l,k}. \tag{2.46}$$

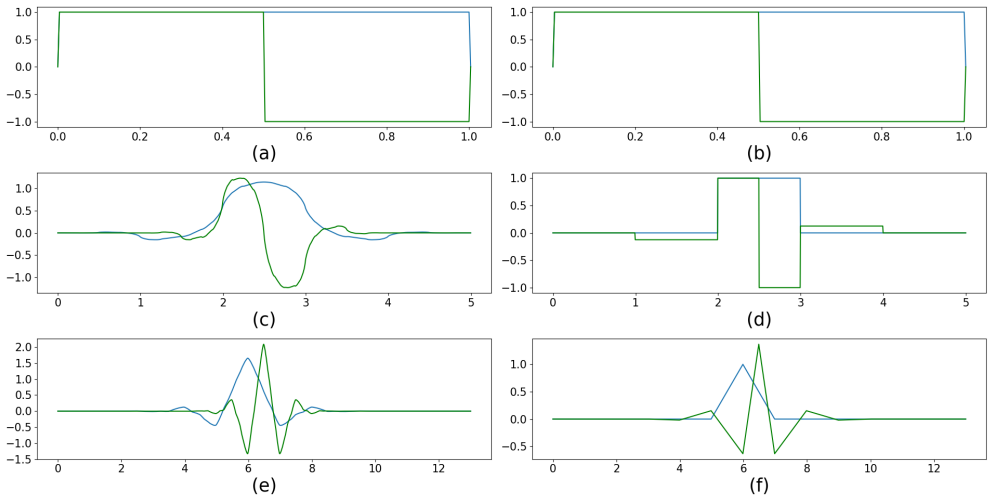

Figure 2.9: Commonly used biorthogonal wavelets. Scaling function in blue and wavelet functions in green. The first number indicates the number of vanishing moments in the deconstruction wavelet and the second number indicates the number of vanishing moments in the reconstruction wavelet. (a) Bior1.1 deconstruction (equivalent to Haar), (b) Bior1.1 reconstruction (equivalent to Haar), (c) Bior 1.3 deconstruction, (d) Bior 1.3 reconstruction, (e) Bior 2.6 deconstruction, and (f) Bior 2.6 reconstruction.

An interesting property of biorthogonal wavelets is that the deconstruction and reconstruction wavelets can be swapped. This means that each biorthogonal wavelet set can provide two wavelet deconstruction/construction operations and [41]:

$$f(t) = \sum_{k=-\infty}^{\infty} \langle f(t), \phi_{rec}(t)_{j,k} \rangle \, \phi_{dec}(t)_{j,k} + \sum_{l \leq j} \sum_{k=-\infty}^{\infty} \langle f(t), \psi_{rec}(t)_{l,k} \rangle \, \psi_{dec}(t)_{l,k}. \tag{2.47}$$

An example of some biorthogonal wavelets are shown in Figure 2.9.

2.3.4 Discrete Wavelet Transform

When represented in the wavelet domain, a discretized waveform, $\bar{v}$, will have two sets of coefficients: A set of scaling coefficients for the selected resolution and multiple sets of wavelet coefficients for the selected resolution and all possible higher resolutions. A scaling coefficient at a selected resolution is determined by:

$$\hat{v}_{\phi,k} = \sum_{n=0}^{M-1} \phi_{j,k,n} \bar{v}_n, \tag{2.48}$$

where $\phi_{j,k,n}$ is the n^{th} sample of $\phi(t)_{j,k}$ once it is discretized into M samples, and $\hat{v}_{\phi,k}$ is scaling coefficient k of the wavelet domain representation of $\bar{v}$.

A wavelet coefficient at a selected resolution is determined by:

$$\hat{v}_{\psi,j,k} = \sum_{n=0}^{M-1} \psi_{j,k,n} \bar{v}_n, \tag{2.49}$$

where $\psi_{j,k,n}$ is the n^{th} sample of $\psi(t)_{j,k}$ once it is discretized into M samples and $\hat{v}_{\psi,j,k}$ is wavelet coefficient k for resolution j. If the wavelet domain representation of $\bar{v}$ is arranged so that the scaling coefficients are first, Equations (2.48) and (2.49) can be represented in matrix form with j_m indicating the selected scale, or resolution, of the transform; $k_m = \frac{M}{2^j}$ indicating the maximum number of translations possible at the j^{th} scale; and W is an $M \times M$ matrix that performs the wavelet transform.

$$\begin{bmatrix} \hat{v}_{\phi,j_m,0} \\ \hat{v}_{\phi,j_m,1} \\ \vdots \\ \hat{v}_{\phi,j_m,k_m-1} \\ \hat{v}_{\psi,j_m,0} \\ \hat{v}_{\psi,j_m,1} \\ \vdots \\ \hat{v}_{\psi,j_m,k_m-1} \\ \hat{v}_{\psi,j_m-1,0} \\ \hat{v}_{\psi,j_m-1,1} \\ \vdots \\ \hat{v}_{\psi,j_m-1,k_m-1} \\ \vdots \\ \hat{v}_{\psi,1,0} \\ \hat{v}_{\psi,1,1} \\ \vdots \\ \hat{v}_{\psi,1,k_m-1} \end{bmatrix} = \begin{bmatrix} \phi_{j_m,0,0} & \phi_{j_m,0,1} & \cdots & \phi_{j_m,0,M-1} \\ \phi_{j_m,1,0} & \phi_{j_m,1,1} & \cdots & \phi_{j_m,1,M-1} \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{j_m,k_m-1,0} & \phi_{j_m,k_m-1,1} & \cdots & \phi_{j_m,k_m-1,M-1} \\ \psi_{j_m,0,0} & \psi_{j_m,0,1} & \cdots & \psi_{j_m,0,M-1} \\ \psi_{j_m,1,0} & \psi_{j_m,1,1} & \cdots & \psi_{j_m,1,M-1} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{j_m,k_m-1,0} & \psi_{j_m,k_m-1,1} & \cdots & \psi_{j_m,k_m-1,M-1} \\ \psi_{j_m-1,0,0} & \psi_{j_m-1,0,1} & \cdots & \psi_{j_m-1,0,M-1} \\ \psi_{j_m-1,1,0} & \psi_{j_m-1,1,1} & \cdots & \psi_{j_m-1,1,M-1} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{j_m-1,k_m-1,0} & \psi_{j_m-1,k_m-1,1} & \cdots & \psi_{j_m-1,k_m-1,M-1} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{1,0,0} & \psi_{1,0,1} & \cdots & \psi_{1,0,M-1} \\ \psi_{1,1,0} & \psi_{1,1,1} & \cdots & \psi_{1,1,M-1} \\ \vdots & \vdots & \vdots & \vdots \\ \psi_{1,k_m-1,0} & \psi_{1,k_m-1,1} & \cdots & \psi_{1,k_m-1,M-1} \end{bmatrix} \begin{bmatrix} \bar{v}_0 \\ \bar{v}_1 \\ \bar{v}_2 \\ \vdots \\ \bar{v}_{M-1} \end{bmatrix}, \tag{2.50}$$

$$\hat{v} = W\bar{v}, \tag{2.51}$$

The inverse wavelet transform is performed differently depending on the type of wavelet being used. For orthogonal wavelets, each time domain coefficient is calculated as:

$$\bar{v}_n = \left(\sum_{k=0}^{k_m-1} \phi_{j_m,k,n}\hat{v}_{\phi,k} + \sum_{j=1}^{j_m-1} \sum_{k=0}^{k_m-1} \psi_{j,k,n}\hat{v}_{\psi,j,k} \right), \tag{2.52}$$

which, due to the orthogonal property of $\phi(t)$ and $\psi(t)$, is equivalent to:

$$\bar{v} = W^{-1}\hat{v} = W^T\hat{v}, \tag{2.53}$$

where W^{-1} is an $M \times M$ matrix that performs the inverse wavelet transform.

With biorthogonal wavelets, the forward transform matrix W, is formed in the same manner as Equation(2.50) but using the $\phi_{dec}(t)$ and $\psi_{dec}(t)$ functions. The inverse wavelet transform matrix is formed by creating a matrix W_b in the same way as Equation (2.50) but using the $\phi_{rec}(t)$ and $\psi_{rec}(t)$ to form the matrix W_b:

$$\bar{v} = W^{-1}\hat{v} = W_b^T\hat{v}. \tag{2.54}$$

An interesting property of biorthogonal wavelets is that, by swapping the rolls of the deconstruction and reconstruction functions, it is possible to get a completely different wavelet representation (i.e. you get two wavelet representations in 1). Reversing the deconstruction and reconstruction wavelets is sometimes called the reverse biorthogonal wavelet. With reverse biorthogonal wavelets, $\hat{v} = W_b\bar{v}$ and $\hat{v} = W^T\bar{v}$. All equations with wavelets in this thesis, can be extended to biorthogonal wavelets by substituting $W^{-1} = W_b^T$ and to reverse biorthogonal wavelets by substituting $W_b = W$.

With some wavelets, some of the coefficients in some of the translations of $\phi(t)$, $\psi(t)$, $\phi_{dec}(t)$, $\psi_{dec}(t)$, $\phi_{rec}(t)$ and $\psi_{rec}(t)$ can fall outside of the time range of the sampled waveform. When using periodic waveforms, coefficients from past the end of a row (require elements from the next time period) are wrapped around to the beginning of the same row (or wrapped around to the end of the same row if elements from the previous time period are required) in the transform matrix.

2.4 Steady-State Wavelet Analysis (SSWA)

The SSWA method is a periodic steady-state analysis method that is derived by transforming Equation (2.14) to the wavelet domain:

$$\begin{aligned}\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) &= \boldsymbol{W}\bar{\boldsymbol{f}}(\boldsymbol{W}^{-1}\hat{\boldsymbol{v}}^j) = \boldsymbol{W}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j) = \boldsymbol{W}\boldsymbol{G}\boldsymbol{W}^{-1}\hat{\boldsymbol{v}}^j + \boldsymbol{W}\boldsymbol{C}\boldsymbol{D}\boldsymbol{W}^{-1}\hat{\boldsymbol{v}}^j + \boldsymbol{W}\bar{\boldsymbol{i}}(\boldsymbol{W}^{-1}\hat{\boldsymbol{v}}^j) \\ &\quad + \boldsymbol{W}\boldsymbol{D}\bar{\boldsymbol{q}}(\boldsymbol{W}^{-1}\hat{\boldsymbol{v}}^j) + \boldsymbol{W}\boldsymbol{\mathcal{F}}^{-1}\boldsymbol{Y}\boldsymbol{\mathcal{F}}\boldsymbol{W}^{-1}\hat{\boldsymbol{v}}^j - \boldsymbol{W}\bar{\boldsymbol{s}},\end{aligned} \tag{2.55}$$

where the hat is used to denote variables in the wavelet domain, $\hat{\boldsymbol{v}}^j = \boldsymbol{W}\bar{\boldsymbol{v}}^j$, $\boldsymbol{W} = I_N \otimes W$, and $\boldsymbol{W}^{-1} = I_N \otimes W^{-1}$. Frequency defined components can be included via the convolution operation which is still performed in the frequency domain. This requires the forward and reverse Fourier transform to be included at the cost of extra matrix operations. However, the matrix $\boldsymbol{W}\boldsymbol{\mathcal{F}}^{-1}\boldsymbol{Y}\boldsymbol{\mathcal{F}}\boldsymbol{W}^{-1}$only needs to be formed once and the majority of the extra computational overhead of this method is found in the forward and reverse wavelet transforms.

In this thesis, the partial derivative of Equation (2.55) with respect to $\bar{\boldsymbol{v}}^j$ is performed in the time domain and the Newton update for iteration j is given by:

$$\boldsymbol{W}\bar{\boldsymbol{f}}'_{\bar{\boldsymbol{v}}}(\bar{\boldsymbol{v}}^j)\Delta\bar{\boldsymbol{v}}^{j+1} = -\boldsymbol{W}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j). \tag{2.56}$$

Substituting $\Delta\bar{v}^{j+1} = \boldsymbol{W}^{-1}\Delta\hat{v}^{j+1}$ into Equation (2.56) results in:

$$\boldsymbol{W}\bar{\boldsymbol{J}}(\bar{v}^j)\boldsymbol{W}^{-1}\Delta\hat{v}^{j+1} = -\hat{\boldsymbol{f}}(\hat{v}^j), \tag{2.57}$$

$$\hat{\boldsymbol{J}}(\hat{v}^j)\Delta\hat{v}^{j+1} = -\hat{\boldsymbol{f}}(\hat{v}^j), \tag{2.58}$$

where $\bar{\boldsymbol{f}}'_{\bar{v}}(\bar{v}^j) = \bar{\boldsymbol{J}}(\bar{v}^j)$,[I] and $\hat{\boldsymbol{J}}(\hat{v}^j) = \boldsymbol{W}\bar{\boldsymbol{J}}(\bar{v}^j)\boldsymbol{W}^{-1}$ is the Jacobian matrix for the SSWA method. An interesting property of this formulation is that, assuming calculations are exact, the calculated correction vectors for each Newton iteration are equivalent. The only difference is the domain the vectors are represented in.

2.5 LU Decomposition of the Jacobian Matrix

The majority of time required for Newton method to converge is spent solving Equations (2.18) and (2.58). Solution of Equations (2.18) and (2.58) is achieved by using LU decomposition in this research. LU decomposition involves decomposing a matrix, A, into a set of factors of the form [42–44]:

$$L_f U_f = PAQ, \tag{2.59}$$

where L_f is a lower triangular matrix (or lower trapezoidal for a tall rectangular A) with a unitary diagonal, U_f is an upper triangular matrix, P is a row permutation matrix, and Q is a column permutation matrix. Each row and column in P and Q contain a single non-zero entry that is equal to 1 and: $PP^T = QQ^T = I$ where I is an identity matrix. These factors, once substituted into a system of equations of the form $Ax = d$, yield:

$$\begin{aligned} (PAQ)Q^T x &= Pd \\ L_f U_f (Q^T x) &= Pd, \end{aligned} \tag{2.60}$$

The solution to the unknown vector x can then be found with forward substitution to solve for y_f in

$$L_f y_f = Pd, \tag{2.61}$$

and backward substitution to solve for x with[II]:

$$U_f(Q^T x) = y_f. \tag{2.62}$$

Solving Equations (2.61) and (2.62) has a computational complexity of order $\mathcal{O}(n_c^2)$ when A is dense where n_c is equal to the number of columns in L_f[44][III]. The factorization of A is $\mathcal{O}(n_c^3)$ when A is dense

[I] From Equation (2.15): $\bar{\boldsymbol{f}}'(\boldsymbol{W}^{-1}\hat{v}^j) = \bar{\boldsymbol{f}}'(\bar{v}^j) = \bar{\boldsymbol{J}}(\bar{v}^j)$

[II] Note: This is achieved by solving for $Q^T x$ with backward substitution and then applying Q to the result to get x.

[III] Solving for the i^{th} element in y_f in Equation (2.61) requires 1 division, $i-1$ multiplication operations, and $i-1$ subtraction operations for a total of $\sum_{i=1}^{n_c} 1 + 2(i-1)$. Using the relation $\sum_{i=1}^{n_c} 1 = n_c$ and $\sum_{i=1}^{n_c} i = \frac{n_c(n_c+1)}{2}$,the total number of operations required for forward substitution is: $\sum_{i=1}^{n_c} 1 + 2(i-1) = \sum_{i=1}^{n_c} 2i - 1 = \frac{2n_c(n_c+1)}{2} - n_c = n_c^2 = \mathcal{O}(n_c^2)$. The number of operations for backward substitution is the same.

[43, 44].

The calculation of the LU factors, which is known as numerical factorization, is performed by starting at the first row and column of the matrix and using Gauss elimination to eliminate all elements in the first column (active column) below the first row (active row). Then the process continues with the second row of the second column as the active row and column, then the third row of the third column as the active row and column, and so on for all columns. There are three operations that can be performed when eliminating the elements below the active row:

1. Two rows can be swapped. These swaps are recorded in the P matrix.

2. Two columns can be swapped. These swaps are recorded in the Q matrix.

3. A row can be multiplied by a constant and then subtracted from another row.

When working with sparse matrices, the columns are normally swapped before performing LU decomposition[44] so only rows are swapped during the elimination process. Calculating the LU factors with only row swapping, or row pivoting, is known as factorization with partial pivoting[I]. Row pivoting is performed to maintain numerical stability [43, 45]. One method for selection of a row to perform pivoting is to look at all elements in the active column and selecting the row with the maximum amplitude element (in the active column) to swap with the active row. Once the pivoting, if required, is performed the elimination is performed on each of the rows below the active row. A simple LU factorization algorithm with row pivoting is shown in Algorithm 2.1.

Performing elimination on the i^{th} row on the F matrix in Algorithm 2.1 requires 1 division, $n_c - i$ multiplications, and $n_c - i$ subtractions over $n_r - i$ rows for a total of

$$\sum_{i=1}^{n_c} (n_r - i)\left[1 + 2\left(n_c - i\right)\right] = \frac{1}{3}\left(-n_c^3 + 3n_r n_c^2 - \frac{3}{2}n_c^2 - \frac{1}{2}n_c\right) \tag{2.63}$$

operations while the pivot step requires $n_r - i$ compare operations, n_c element swap operations for the F matrix, and n_c element swap operations for the P matrix for a total of

$$\sum_{i=1}^{n_c} n_r - i + 2n_c = \frac{3}{2}n_c^2 + n_r n_c - \frac{1}{2}n_c \tag{2.64}$$

operations.

[I]Factorization with full pivoting is when both row and column swapping is performed.

Algorithm 2.1 Simple LU factorization algorithm for an $n_r \times n_c$ matrix.

$F \leftarrow$ copy of original $n_r \times n_c$ matrix to be factored. L_f and U_f factors are stored in this matrix as they are found.

$P \leftarrow$ is the row permutation matrix. Initial value is an $n_r \times n_r$ identity matrix.

$i \leftarrow 1$

while $i <= n_c$ **do**

  $g_{max} \leftarrow$ index of row in column i of F that has maximum amplitude element.

  if $i \neq g_{max}$ **then**

    Swap row i with row g_{max} in the P matrix.

    Swap row i with row g_{max} in the F matrix.

  end if

  $j \leftarrow i + 1$

  while $j <= n_r$ **do**

    $F_{j,i} = F_{j,i}/F_{i,i}$ // Store elements for in L matrix portion of F.

    $k \leftarrow i + 1$

    while $k <= n_c$ **do**

      $F_{j,k} = F_{j,k} - F_{j,i} \times F_{i,k}$

      $k \leftarrow k + 1$

    end while

    $j \leftarrow j + 1$

  end while

  $i \leftarrow i + 1$

end while

To obtain the L_f and U_f matrices from F:

- All elements at and above the diagonal in the first n_c rows of F become the upper triangular portion of U_f.
- All elements below the diagonal in the first n_c rows of F become the lower triangular ($n_r = n_c$) or trapezoidal ($n_r > n_c$) portion of L_f and all diagonal elements in first n_c rows of L_f are equal to 1.

When $n_r = n_c$, Equation (2.63) simplifies to $2n_c^3 - \frac{3}{2}n_c^2 - \frac{1}{2}n_c$ which is of order $\mathcal{O}(n_c^3)$ and Equation (2.64) simplifies to $\frac{5}{2}n_c^2 - \frac{1}{2}n_c$ which is of order $\mathcal{O}(n_c^2)$. If F is a tall rectangular matrix with $n_r > n_c$ and $n_r = pn_c$ where $1 < p$, then Equation (2.63) simplifies to $\frac{1}{3}\left(-n_c^3 + 3pn_c^3 - \frac{3}{2}n_c^2 - \frac{1}{2}n_c\right)$ which, since

$p = \frac{n_r}{n_c}$, is of order $\mathcal{O}(n_r n_c^2)$[I]. Equation (2.64) simplifies to $\frac{3}{2}n_c^2 + pn_c^2 - \frac{1}{2}n_c$ and is of order $\mathcal{O}(n_c^2)$. Thus, with a tall rectangular matrix, the number of columns is the determining factor for the computational complexity of LU factorization of a dense matrix.

When performing SSTD and SSWA on electric circuits, the Jacobian matrices are normally sparse. Since the LU decomposition in Algorithm 2.1 will operate on all elements, regardless of if they are non-zero or not, it is an inefficient method of finding the L and U factors of a sparse matrix. Therefore, it is necessary to utilize a method that operates only on the non-zero entries of the Jacobian matrix. Methods that perform LU factorization on sparse matrices operate by partitioning the matrix into smaller, sparse sub-matrices based on the positions of the non-zero elements in the original sparse matrix [44–48]. These methods typically include the following phases[44–48]:

- Matrix ordering to reduce fill-in: During this phase, the matrix columns are re-ordered in a way that is expected to provide the sparsest L_f and U_f matrices (the Q matrix is selected during this phase) and provide an upper bound on fill-in. Fill-in occurs when new non-zero entries are introduced into the L_f and U_f matrices that were not present in the original A matrix. These fill-ins increase the amount of computational time required for factorization since they will require extra time to compute their values as well as require more allocated memory to store.

- Symbolic factorization: During this stage, the upper bound to the fill-in associated with the LU decomposition of the matrix and the memory used for the factorization step is estimated and allocated for use with the numerical factorization.

- Search for dense sub-matrices and numerical factorization: This is the step where the work is performed. The matrix is sub-divided into multiple dense blocks which are factorized to generate the L_f and U_f matrices. The row order, provided by the P matrix, is also refined in this step.

This research utilizes the UMFPack routines in SuiteSparse [48, 49]. UMFPack performs numerical factorization by forming smaller, dense, matrices known as frontal matrices out of columns and rows in the original matrix and performing the factorization on the frontal matrices. The size of the frontal matrices varies and is dependant on the number of non-zero elements in the rows and columns used to create the frontal matrix at each stage of factorization[48, 49]. The number of iterations required in the factorization depends on how many columns are utilized in the frontal matrices. At worst, one column is added to each frontal matrix each iteration. This would mean that elimination would be performed one

[I]Since p is always greater than 1 (more rows than columns), $3pn_c^3$ is the dominant term regardless of the value of p so the complexity is of order $\mathcal{O}(pn_c^3)$. Substitution of $p = \frac{n_r}{n_c}$ provides the result in terms of n_r and n_c which results in $\mathcal{O}(n_r n_c^2)$.

column at a time just like Algorithm 2.1. However, more than one column can be added at a time and, therefore, it is possible for UMFPack to finish factorization in fewer than n_c iterations.

One of the contributions of this thesis is a method that reduces the columns in the Jacobian matrix at each iteration. Reducing the columns could lead to a potential speedup by reducing the number of iterations required for UMFPack to finish by reducing n_c. Another contribution of this thesis is a method that reduces the number of non-zero entries in the Jacobian matrix. Since the pre-ordering is selected to minimize fill-in, a matrix with fewer non-zero entries could require less memory resources and computational time to factorize. However, the method utilized to reduce the number of non-zero entries in the Jacobian matrix can lead to potential problems with respect to fill-in since the sparsity pattern (the positions of the non-zero elements in the Jacobian matrix) can be altered.

2.6 Chapter Summary

In this chapter, the reader was introduced to the nodal analysis used in this work to form the system of circuit equations, how these equations are used for periodic steady-state analysis, as well as a brief introduction to wavelets and how they are used for wavelet domain periodic steady-state analysis. Also included was a brief introduction to dense and sparse LU factorization and the UMFPack algorithm that is utilized for LU factorization in this research. The next chapter will discuss other work that has been done in the area of utilizing wavelets to increase the efficiency of steady-state analysis. These methods operate by either reducing the size of the problem to be solved using Newton method or by reducing the number of non-zero elements in the Jacobian matrix. Also included in the chapter summary is a comparison of the benefits and drawbacks of these methods compared with the contributions of this thesis.

Chapter 3

Literature Review

Wavelet domain analysis is not a new idea in circuit simulation. Some proposed applications are: transient, steady-state, and sensitivity analysis [9, 11, 16, 17, 50–53]. Wavelets have been utilized to improve the efficiency of steady-state analysis by:

1. Reduction of problem size by taking advantage of the sparse representation of signals in the wavelet domain. [8–11]

2. Reduction in the size of the system of equations by selecting the level of wavelet transform to be used during the circuit simulation. [12–15]

3. Increasing the sparsity of the Jacobian matrix. [16, 17]

This section will discuss some of the research performed in each of these three methods of improving efficiency. Note that, where possible, the same notation and symbols used in Chapter 2 were used in this chapter and, as a result, some of the formula symbols differ from the original sources.

3.1 Reduction of Problem Size

One way of taking advantage of the ability of wavelets to sparsely represent signals is proposed by the authors of [8, 10, 11, 54]. The method is applicable to both transient analysis [9] and periodic steady-state analysis (PSSA) [8]. This method utilizes a specifically designed spline wavelet that has an explicit formulation and allows for derivatives and integrals to be performed directly in the wavelet domain [10, 54].

The circuit equations are formulated using nodal analysis to obtain[8, 10]:

$$\frac{d}{dt}q(v(t)) + i(v(t)) - s(t) = 0, \tag{3.1}$$

where $v(t) : \mathbb{R} \to \mathbb{R}^N$ denotes the unknown voltages and currents, $i(v(t))$ describes the static contributions from resistors, diodes, transistors, etc., $q(v(t))$ contains charges and fluxes from capacitors and inductors, and $s(t)$ contains the independent sources. The solution to Equation (3.1) is found using a Petrov-Galerkin approach. The unknown solution is discretized in terms of the wavelet basis functions. That is, the spline wavelet is used to discretized $v(t)$ in Equation (2.8) into $n+1$ coefficients[8, 10, 54]:

$$v(t) = \sum_{k=0}^{n} \hat{v}_k \Psi_k(t), \tag{3.2}$$

where $\hat{v}_k \in \mathbb{R}^N$ and $\Psi_k(t)$ is the k^{th} periodic basis function for the spline wavelet. Equation (3.1) is then integrated over a chosen set of n sub intervals $[\tau_{l-1}, \tau_l]$[I] to define:

$$f_l(\hat{v}_0, \hat{v}_1, \ldots, \hat{v}_n) = q(v(\tau_l)) - q(v(\tau_{l-1})) + \int_{\tau_{l-1}}^{\tau_l} i(v(t)) + s(t)dt, \tag{3.3}$$

for $l = 1, \ldots, n$ and solved using Newton method.

This method begins with a set of equally spaced knots. Each iteration, Equation (3.3) is solved using Newton method. Then each interval is checked and equally spaced knots are added to regions where sharp changes are detected by using a tunable refinement rate parameter and removed from smooth regions by thresholding knots with corresponding wavelet coefficients that are below a selected threshold. The new representation with the adjusted knots is then used as the initial guess for Newton method to solve Equation (3.3) again until the square norm of the difference between the result for Equation (3.3) on the current iteration and the result for Equation (3.3) on the previous iteration is sufficiently small. This method was extended to handle multirate circuit simulations as well.

The study in [10] was performed using transient simulations. The wavelet transient algorithm achieved results that agreed with a time domain transient analysis. Although the wavelet method had a lower number of spline knots than the time domain transient analysis had time steps, the computation time was nearly always higher with the wavelet transient algorithm. In [11], the method was extended to find the periodic steady-state solution to multirate circuit problems. The results show that the wavelet PSSA algorithm is much faster than utilizing time domain transient analysis to find the steady-state solution for the test cases. The results of [8] show the operation of the algorithm simulating a diode rectifier and several multirate circuit problems for PSSA.

The algorithm is successful in managing the number of knots used to describe the nodal variable waveforms. Early iterations of the multirate simulations tended to require more knots to describe the waveforms and required less as the simulation converged. For the diode rectifier, the number of knots increased as the simulation converged and then was decreased for the final iterations where no new knots were added but many were removed for the last round of Newton method.

[I] These sub-intervals are determined by the knots in the splines.

An advantage of this method is that, since the previous iterations nodal variable vector is used as the initial guess for the internal Newton iterations, the Newton method tends to converge very quickly since it starts close to its final solution.

3.2 Reduction in Size of the System of Equations

It is possible to utilize wavelets to reduce the size of the system of equations by selecting the level of transform used during the simulation. These methods operate by starting with a low resolution wavelet transform and only calculating the scaling coefficients for that resolution. Leaving all the wavelet subspace coefficients at zero allows for a low resolution estimate of the solution to be calculated. This estimate can then be improved by adding elements from the higher level wavelet subspaces to fill in the missing details and refine the rough estimate.This approach has been proposed for both transient [12, 13] and steady-state analysis [14, 15].

Transient analysis was used in [12] with the Haar wavelet to develop a method that takes advantage of the properties of the block pulse function to derive derivative and integral operators. These operations are used to form matrices that perform the integrals and derivatives:

$$Q_H = WQ_BW^{-1}, \tag{3.4}$$

$$Q_H^{-1} = WQ_B^{-1}W^{-1}, \tag{3.5}$$

where Q_B and Q_B^{-1} are $M \times M$ matrices that perform the integration and differentiation operations, respectively, on a vector with M samples and are defined in [12, 13]. This method does not use nodal analysis but instead uses Kirchhoff's Current Law and Kirchhoff's Voltage Law to directly find the voltage or current in a component. The voltages and currents are derived for resistors, inductors, and capacitors:

$$\hat{v}_r = R_w\hat{i}_r, \tag{3.6}$$

$$\hat{v}_l = Q_H^{-1}L_w(\hat{i}_l - \hat{i}_{l,o}), \tag{3.7}$$

$$\hat{i}_l = \hat{i}_{l,o} + L_w^{-1}Q_H\hat{v}_l, \tag{3.8}$$

$$\hat{v}_c = \hat{v}_{c,o} + C_w^{-1}Q_H\hat{i}_c, \tag{3.9}$$

$$\hat{i}_c = C_wQ_H^{-1}(\hat{v}_c - \hat{v}_{c,o}), \tag{3.10}$$

where $\hat{v}_r$ and $\hat{i}_r$ are the wavelet domain resistor voltage and current respectively, $\hat{v}_l$ and $\hat{i}_l$ are the wavelet domain inductor voltage and current respectively, $\hat{v}_c$ and $\hat{i}_c$ are the wavelet domain capacitor voltage and current respectively, and $\hat{v}_{c,o}$ and $\hat{i}_{l,o}$ are the wavelet domain initial capacitor voltage and inductor current respectively. R_w, L_w, and C_w are $M \times M$ matrices that represent the resistance, inductance, and capacitances of a given resistor, inductor, or capacitor and are defined in [12, 13]. This derivation

is extended to handle coupled inductors and non-linear time variant circuit elements as well. For more details on these elements, the reader is encouraged to review [12, 13].

A simulation with this method begins by selecting an element. Kirchhoff's Voltage Law and Kirchhoff's Current Law is then used to solve for either the current or voltage of the selected component. The derived equation is then solved at a low resolution. The simulation time period is divided into smaller sub-intervals and each is analyzed either manually [12], or automatically [13] and the local resolution of the solution in sub-intervals where singular points are detected is then increased while the other sub-intervals are left at the original resolution and the derived equation is solved again. The results show that there is an advantage to using the mixed resolution method as opposed to using the higher resolution for the entire time period. However, this method calculates one voltage or current variable at a time and, if more are required, a new analysis has to be prepared and run for each.

Another approach, proposed in [14], utilizes Chebyshev polynomials that are defined on the interval $[-1, 1]$ to create an orthogonal wavelet basis for the steady-state analysis of power electronics circuits. This method then approximates the solution to the circuit equations by selecting a desired resolution which is achieved by varying the level of the wavelet transform. In this method, power electronic circuits are represented by:

$$\frac{d}{dt}v(t) = A(t)v(t) + s(t), \tag{3.11}$$

where $A(t)$ is an $N \times N$ time varying matrix that represents the circuit. Utilizing the developed Chebyshev polynomial wavelet, $v(t)$ can be expressed with[14]:

$$v(t)_i = K_i^T \Psi_{ch}(t), \qquad \text{for } i = 1, \ldots, N \tag{3.12}$$

where $\Psi_{ch}(t)$ is a wavelet basis of level l of dimension $2^{l+1} + 1$ constructed with Chebyshev polynomials and $K_i^T = \left[k_{i,0}, k_{i,1}, \cdots, k_{i,2^{l+1}}\right]$ is an unknown coefficient vector of dimension $2^{l+1} + 1$. Equation (3.11) can be expressed as:

$$KD\Psi_{ch}(t) = A(t)K\Psi_{ch}(t) + s(t), \tag{3.13}$$

where D is a matrix that represents the derivative operation and

$$K = \begin{bmatrix} K_1^T \\ K_2^T \\ \vdots \\ K_N^T \end{bmatrix}. \tag{3.14}$$

Re-arranging K into a vector as $K_v = \left[K_1^T, K_2^T, \ldots, K_N^T\right]^T$ allows Equation (3.12) to be re-written as:

$$F(t)K_v = s(t), \tag{3.15}$$

$$F(t) = \begin{bmatrix} a_{11}\Psi_{ch}^T(t) & \cdots & a_{1i}\Psi_{ch}^T(t) & \cdots & a_{1N}\Psi_{ch}^T(t) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1}\Psi_{ch}^T(t) & \cdots & a_{ii}\Psi_{ch}^T(t) & \cdots & a_{iN}\Psi_{ch}^T(t) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{N1}\Psi_{ch}^T(t) & \cdots & a_{Ni}\Psi_{ch}^T(t) & \cdots & a_{NN}\Psi_{ch}^T(t) \end{bmatrix} - \Psi_{ch}D, \tag{3.16}$$

$$\Psi_{ch}D = \begin{bmatrix} \Psi_{ch}^T(t)D^T & & & & \\ & \ddots & & & \\ & & \Psi_{ch}^T(t)D^T & & \\ & & & \ddots & \\ & & & & \Psi_{ch}^T(t)D^T \end{bmatrix}. \tag{3.17}$$

Unknown vector K_v requires $(2^{l+1}+1)N$ equations to solve[I]. The periodic boundary condition, which requires $x(1) = x(-1)$, provides N of the equations [14]. The other $2^{l+1}N$ equations are provided by the interpolation points of the Chebyshev wavelets.

$$\begin{bmatrix} \hat{F}_1 \\ \hat{F}_2 \end{bmatrix} K_v = \begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix}, \tag{3.18}$$

where $\hat{F}_1$ and $\hat{s}_1$ have N rows and $\left(2^{l+1}+1\right)N$ columns and are given by

$$\hat{F}_1 = \begin{bmatrix} \Psi_{chr} & & & \\ & \Psi_{chr} & & \\ & & \Psi_{chr} & \\ & & & \Psi_{chr} \end{bmatrix}, \tag{3.19}$$

$$\Psi_{chr} = \left[\Psi_{ch}(1) - \Psi_{ch}(-1)\right]^T. \tag{3.20}$$

Due to the periodic boundary condition, $\left[\Psi_{ch}(1) - \Psi_{ch}(-1)\right]^T K_i = 0$ and

$$\hat{s}_1 = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{3.21}$$

[I] K_v has $(2^{l+1}+1)N$ elements.

$\hat{F}_2$ and $\hat{s}_2$ have $2^{l+1}N$ rows and $\left(2^{l+1}+1\right)N$ columns and are given by

$$\hat{F}_2 = \begin{bmatrix} F(\xi_1) \\ F(\xi_2) \\ \vdots \\ F(\xi_{2^{n+1}}) \end{bmatrix}, \tag{3.22}$$

$$\hat{s}_2 = \begin{bmatrix} -s(\xi_1) \\ -s(\xi_2) \\ \vdots \\ -s(\xi_{2^{n+1}}) \end{bmatrix}, \tag{3.23}$$

where ξ_i are the interpolation points of the wavelets...

With the formulation in [14, 15], increasing the level of transform will increase the resolution of the wavelet domain representation by including more wavelet coefficients[I]. Thus, increasing the level of wavelet transform will increase the size of $\hat{F}_2$ and $\hat{s}_2$ by including more wavelet coefficients in the calculations and lower level transforms will lead to smaller systems of equations. The interpolation points can be selected to be anywhere in the interval $[-1, 1]$[II] but, a poor choice can lead to numerical instability and the authors of [14] recommend using the interpolation points of the wavelets.

It is noted in [14] that higher level wavelets are only required at times where there are high variations, such as switching instants, in the waveforms. Thus, by using higher level wavelets for parts of the solution waveform that show high variability and lower level wavelets for the parts of the solution waveform that show low variability, it is possible to represent the solution with an over all lower number of wavelet coefficients. This is achieved by partitioning $\Psi_{ch}(t)$ into two parts: Ψ_{ch1} which represents the wavelets that are selected for use in the approximation of the solution and Ψ_{ch2} which represents the un-chosen wavelets. This partitioning results in:

$$\Psi_p = P\Psi_{ch}(t) = \begin{bmatrix} \Psi_{ch1} \\ \Psi_{ch2} \end{bmatrix}, \tag{3.24}$$

where P is a permutation matrix which sorts the chosen and un-chosen wavelets. The differentiation matrix must also be sorted to accommodate this new partitioned scheme:

$$D = PDP^{-1} = \begin{bmatrix} D_1 \\ D_2 \end{bmatrix}. \tag{3.25}$$

[I] Remember, for the formulation in this thesis, increasing the level will lower the resolution of the scaling subspace representation.

[II] This interval was selected because the Chebyshev polynomials were defined on this interval.

The partitioning described above results in an $\hat{F}_1$ and $\hat{s}_1$ which each have N rows and an $\hat{F}_2$ and $\hat{s}_2$ which each have $(M-1)N$ rows where M is the number of wavelets that have been chosen for the approximation (i.e. the number of elements in Ψ_{ch1}). For a more detailed description of the derivation and resulting $\hat{F}_1$, $\hat{F}_2$, $\hat{s}_1$, and $\hat{s}_2$ see [14]. Low level wavelets are automatically included in Ψ_{ch1} for waveforms that contain a substantial amount of low-frequency components while higher level transforms are only included for waveforms that have high-frequency components. The higher frequency wavelets are selected based on distance from a switching point with an empirical rule of $\frac{1}{10}$ to $\frac{1}{2}$ of the interval being a recommended range.

The results presented in [14] show that it is possible to obtain approximations of the solution waveforms using fewer coefficients than a Fourier series approximation would require to achieve the same level approximation error. However, in cases where there are few high-frequency details around switching instants, the Fourier approximation requires significantly more coefficients to reduce the average error to a level that is below that of the wavelets. Even with extra coefficients, the maximum error was much higher when using the Fourier approximation. With both the wavelet approximation method and the improved method, the accuracy of the approximation is determined by how many low level wavelets are included in Ψ_{ch1} at the beginning of the simulation and also the selected distance threshold used to determine which wavelets to add to Ψ_{ch1} while the simulation is run. The selection of how many low level wavelets to include at the beginning of the simulation, as well as the selected distance threshold, can have a significant effect on the error and are left up to the designer to select prior to running a simulation.

In [15], the method is further improved by adding a convolution of $\hat{F}_2$ and $\hat{s}_2$ with the wavelet basis functions. This makes the exact locations of the data points unimportant in the computations and provides a more robust and accurate solution to Equation (3.18). However, this method still requires a selection of wavelet levels to use in the simulations. Simulations were run for the same circuits as used in [14] in order to provide a comparison between the proposed and original, un-partitioned, methods. The partitioned method presented in [14] is not compared and errors are calculated using SPICE simulation results as the baseline. The results show that the convolution method does provide greater accuracy than the original method for any given selected set of wavelet levels. This error can be significant around switching points if the number of wavelet levels utilized is too low. This could be problematic since it is left to the designer to select how many levels to include prior to the start of the simulation. Simulation times were also explored in this paper with simulation speeds being faster with the improved method. Interestingly, based on the results presented, the partitioned method in [14] appears to attain lower measured errors than the improved method in [15]. However, since there is no direct comparison with the partitioned method it is unclear how the computational speed of the two methods compare.

3.3 Increasing the Sparsity of the Jacobian Matrix

Wavelet domain Jacobian matrix sparsity can be increased by either thresholding (setting to zero) low amplitude entries in the Jacobian matrix itself or in one of the matrices that the Jacobian is composed of. For example, the authors in [17] presented a method that used a pre-selected threshold for thresholding low amplitude entries in the wavelet domain version of the convolution matrix used for frequency defined components. The behaviour of non-linear circuits with frequency defined components can be described with:

$$(G \otimes I_M)\,\bar{v}_s + (C \otimes I_M)\,D_s\bar{v}_s + \bar{f}_s(\bar{v}_s) + \Upsilon_s\bar{v}_s - \bar{s}_s = \bar{\mathbf{0}}, \tag{3.26}$$

where $\bar{f}_s(\bar{v})$ is the discretized NM element vector representing nonlinear components arranged so all nodal variables for a given sample are grouped together[I], D_s is an $NM \times NM$ matrix that represents the derivative operation and is defined in [53], Υ_s is an $NM \times NM$ convolution matrix used for frequency defined components and is defined in [17], $\bar{v}_s$ is an NM element vector of nodal variables (node voltages, branch currents, inductor fluxes, capacitor charges, etc.) arranged so all nodal variables for a given sample are grouped together, $\bar{s}_s$ is a vector of independent sources arranged so all nodal variables for a given sample are grouped together.

Transforming the equations into the wavelet domain results in:

$$\Phi(W_s\bar{v}_s) = (G + CD + \Xi)\,W_s\bar{v}_s + W_s\bar{f}_s(\bar{v}_s) - W_s\bar{s}_s = \bar{\mathbf{0}}, \tag{3.27}$$

where W_s and W_s^{-1} are $NM \times NM$ matrices that perform the forward and reverse wavelet transforms as defined in [17] and $\Xi = W_s\Upsilon_s W_s^{-1}$. The Jacobian matrix associated with Equation (3.27) is defined in [17] as:

$$J(W_s\bar{v}_s) = (G + CD + \Xi) + W_s\left[\frac{\partial f_k}{\partial x_l}\right]W_s^{-1} = \bar{\mathbf{0}}. \tag{3.28}$$

Representation of Ξ tends to be dense and this increase in density increases the computational cost of factoring the Jacobian matrix when applying Newton method to Equation (3.27). Thus, the authors of [17] proposed thresholding the low amplitude elements from Ξ to decrease the Jacobian matrix density. This was achieved by checking each of the $N \times N$ blocks that make up $J(W_s\bar{v}_s)$ and thresholding all elements that are below the threshold (i.e. setting them to zero) which is calculated as:

$$h_s = \gamma \cdot max\left|\Xi_{i,j}\right|, \tag{3.29}$$

where $0 < \gamma < 1$ is a thresholding parameter used to set the threshold and $\Xi_{i,j}$ is the $(i,j)^{\text{th}}$ block of Ξ. Since Ξ is thresholded before the simulation begins, there is a trade off between the sparsity and accuracy of the thresholded convolution operation and there is a risk of causing an error in the final result

[I] All variables for sample 1 are grouped together followed by the variables for sample 2, etc.

of the simulation if the threshold, which has to be selected by the designer running the simulation, is too aggressive. Additionally, since thresholding is performed at the beginning of the simulation, Υ_s has to be invariant of $\bar{v}_s$ [17].

The results of this paper show that the sparsity of the wavelet domain Jacobian matrices are much greater, once thresholding is applied, than the Jacobian matrices of an equivalent frequency domain analysis[I]. However, since Ξ is used in the formulation of the system of equations, thresholding elements from it leads the introduction of a small error into the final result of the thresholded simulations compared to the non-thresholded simulations.

3.4 Review of Wavelet Selection Methods

There are many ways of measuring the performance of a wavelet and usually the performance metrics are problem dependent. For example, in [19], it was shown that 8 coefficient Daubechies wavelets provided the best performance, out of the wavelets being compared, for compression and denoising of electrocardiogram (ECG) signals. Denoising of the signal was achieved by thresholding low amplitude coefficients from the noisy signal. The determination of best performance was based on how sparsely the signal was represented by the wavelets being compared, how much Root Mean Squared (RMS) error was introduced when low amplitude coefficients were thresholded, and how well the peaks of the denoised signal were preserved.

In [20] several wavelet families were analyzed and compared for use with measurement of steady-state harmonic distortion of power systems. Since the authors were interested in the third and fifth harmonic components of the signal, they selected a five level transform. The scaling coefficients at this level would contain the fundamental component while the fifth and fourth detail levels would contain the third and fifth harmonic components of the original waveforms. The energy of the wavelet coefficients at each level was compared with the calculated energy of each harmonic component of the waveforms studied and the one with the least deviation between energy at each level and energy of each harmonic component was considered the best. In this case, the Coiflet family of wavelets was found to be most suitable.

In [21] a study was performed on using wavelets with partial discharge (PD) detection. The authors discussed a method for automatically selecting an appropriate mother wavelet by maximizing the correlation coefficient between the signal being represented and the mother wavelet (this method was developed by authors in [22]). This indicates that the sparsest wavelet domain representations of a signal occur with Mother wavelets that exhibit similar behaviour to the signal. The paper also tests automatic thresholding methods that are commonly used for removing noise. Since the mother wavelet is highly correlated with

[I]The frequency domain analysis was performed by substituting matrices that perform the forward and reverse Fourier transform operations for W_s and W_s^{-1}.

the PD pulse signal being detected, most of the energy of the PD pulse signals in the wavelet domain will be concentrated on a few wavelet subspaces (i.e. the signal is sparse) while the uncorrelated noise will be spread out over all subspaces (i.e. is not sparse) and the noise can be removed by thresholding off the low amplitude wavelet coefficients which leaves only the high amplitude wavelet coefficients that correspond to the PD pulse signal being detected.

In [23], a comparison analysis of the sparsity of the Jacobian and derivative matrices between the biorthogonal cubic spline wavelet in [55] and the Daubechies wavelet is presented. The authors [23] used a formulation that was similar to that used in [17] but did not consider frequency defined components in their circuit equation formulation. It is shown that the biorthogonal cubic spline wavelet led to higher sparsity Jacobian and derivative matrices in their analysis. However, it is unclear what effect the inclusion of frequency defined components would have on the Jacobian and derivative matrices.

3.5 Chapter Summary

In this chapter is a review of previous work performed on taking advantage of the sparse representations that are possible with wavelets to increase the efficiency of periodic steady-state analysis. One topic of research focused on utilizing wavelets to reduce the density of the Jacobian matrices by thresholding the convolution matrix used with frequency defined components. The convolution matrix is represented in the wavelet domain by including the forward and reverse transform matrices when the convolution matrix is formed. Other works focused on utilizing the properties of a specific wavelet to reduce the size of the system of equations by re-formulating the problem to calculate a sparse estimate of the solution to the system of equations. The volume of work in this area is very limited. Based on the results of the reviewed works as well as the results of this research, there is an advantage to SSWA in some cases. However, wither or not wavelet domain steady-state analysis provides a computational advantage, as well as the selection of the most suitable wavelet, is largely problem dependent. Therefore it is difficult to create a one size fits all method which makes progress in this area of research slow. However, there is still much room for further developments and research on this interesting topic. The following two chapters will discuss the contributions of this thesis into ways to increase the efficiency of calculating the Newton update vectors during each iteration of Newton method.

Presented in Chapter 4 is a study on the effect wavelet selection has on Jacobian matrix and nodal variable vector sparsity. This study is an important requirement for further work since there were no previous studies to determine which wavelet, if any, tends to provide the sparsest Jacobian matrix and nodal variable vectors. Also presented and tested is an adaptive Jacobian matrix thresholding algorithm. This algorithm was developed to address the issue of selection of an appropriate threshold by automatically adjusting it each iteration of Newton method. The threshold is dependant on error introduced into the

Newton update vectors each iteration and, unlike the method described in Section 3.3, will not introduce an error into the final result of the simulation.

Presented in Chapter 5 is a method that reduces the number of columns in the Jacobian matrix that are used in the calculation of the Newton update vectors. This method was developed to allow for the use of any wavelet basis. This method is modified to allow for the wavelets to be adaptively selected each iteration. Utilization of a different wavelet basis for every nodal variable is also explored.

Chapter 4

Effect of Wavelet Selection on Periodic Steady-State Analysis

An important consideration when selecting a wavelet is its effect on SSWA. Since the SSTD and SSWA formulations in this thesis are equivalent (see Section 2.4), the wavelet choice has no effect on the convergence of Newton method. The main effect of the wavelet choice is in the density of the nodal variable vectors and the structure, and sparsity, of the Jacobian matrix. However, selection of an optimum wavelet basis function for circuit simulation is not a straight forward task. The nodal variable waveforms for circuit simulations can potentially have completely different shapes. This applies to nodal waveforms of differing circuits as well as the waveforms for each node within the same circuit. Most of the nodal waveforms are also unknown until they are calculated. A designer could select a wavelet based on the shape of the input and expected output waveforms but this does not guarantee the selected wavelet will provide the sparsest representations of the other nodes in the circuit. For this reason, a study on the effect that wavelets selected from several families have on the sparsity of the nodal variable vectors and Jacobian matrices was performed to determine which wavelet, if any, provides the sparsest vectors and matrices.

4.1 Adaptive Jacobian Thresholding

The study presented in this section, and published in [18], explores the effect that wavelets selected from several families have on the sparsity of the nodal variable vectors and Jacobian matrices with five test circuits. A method that performs adaptive thresholding, with an automatically selected threshold, on the entire Jacobian matrix each iteration is also presented and tested. Unlike the method in [53], this method does not run the risk of causing a deviation between the un-thresholded and thresholded simulations since any error caused by thresholding the Jacobian matrix will be corrected during a later Newton iteration. Additionally, since the Jacobian matrix is thresholded at each iteration, the convolution

matrix can vary with $\bar{v}$. Another advantage of the automatic thresholding method is that a designer does not have to determine an ideal threshold for use with each circuit problem. However, this method has three disadvantages:

1. Extra computational overhead every iteration to perform the thresholding.
2. An error, which is dependent on how aggressive a threshold is used, is introduced to the calculation of $\Delta\hat{\boldsymbol{v}}^{j+1}$.
3. Overaggressive thresholding can lead to additional iterations to converge or even a failure to converge.

In order to overcome the first disadvantage, the thresholding algorithm has to be efficient enough for the overhead of thresholding the Jacobian matrix to be lower than the computational speedup of solving the sparser, thresholded, Jacobian matrix. The adaptive method used in this study is computationally simple. All elements in the Jacobian matrix that have amplitudes below the adaptive threshold (h) are set to zero and control of the threshold is achieved via simple vector and matrix/vector operations.

To overcome the second and third disadvantages, the adaptive method used in this study adapts the threshold to control and minimize the error introduced into the computation of $\Delta\hat{\boldsymbol{v}}^{j+1}$. This is achieved by adjusting the value of h based on the error introduced in the computations due to thresholding the Jacobian matrix. After selecting a small initial value for h, an approximate Newton update ($\Delta\hat{\boldsymbol{v}}_h^{j+1}$) is obtained by using the thresholded Jacobian ($\hat{\boldsymbol{J}}_h$) to express Equation (2.58) as:

$$\hat{\boldsymbol{J}}_h \Delta\hat{\boldsymbol{v}}_h^{j+1} = -\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j). \tag{4.1}$$

The Newton update is given by:

$$\Delta\hat{\boldsymbol{v}}^{j+1} = \Delta\hat{\boldsymbol{v}}_h^{j+1} + \hat{\boldsymbol{\delta}}, \tag{4.2}$$

where $\hat{\boldsymbol{\delta}}$ is a vector that represents the difference between $\Delta\hat{\boldsymbol{v}}^{j+1}$ and $\Delta\hat{\boldsymbol{v}}_h^{j+1}$. Substituting Equation (4.2) into Equation (2.58) yields:

$$\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)(\Delta\hat{\boldsymbol{v}}_h^{j+1} + \hat{\boldsymbol{\delta}}) = -\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j),$$
$$\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\hat{\boldsymbol{\delta}} = -\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) - \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}.$$

The introduced error, $\hat{\boldsymbol{\epsilon}}$, is then given by:

$$\hat{\boldsymbol{\epsilon}} = |\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\hat{\boldsymbol{\delta}}| = |\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) + \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}|, \tag{4.3}$$

where the vertical bars denote the absolute value of all elements in a vector. The values of the individual elements of $\hat{\boldsymbol{\epsilon}}$ should be comparable to the tolerance used to evaluate convergence of the Newton algorithm.

If an element is too large, then h must be reduced, conversely if all elements in $\hat{\boldsymbol{\epsilon}}$ are very small, this indicates that h may be increased. If $\hat{\boldsymbol{\epsilon}}$ is in an acceptable range, h is left unchanged. Each element in $\hat{\boldsymbol{\epsilon}}$ is controlled by comparison with an automatically calculated tolerance:

$$\hat{\boldsymbol{t}} = r_{tol}\text{minimum}\{|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)|, |\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}|\} + a_{tol}\hat{\boldsymbol{u}}, \tag{4.4}$$

where $\hat{\boldsymbol{u}} \in \Re^{NM}$ is a vector with all elements set to 1 and r_{tol} and a_{tol} are selected relative and absolute tolerances. Operation minimum$\{\}$ performs an element wise comparison between $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$ and $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}$ with the result being a vector containing the minimum between the elements of $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$ and $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}$. The relative tolerance ensures that each element in $\hat{\boldsymbol{\epsilon}}$ will have amplitudes that are much lower than the corresponding minimum element between $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}$ and $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$.

Algorithm 4.1 Jacobian matrix adaptive threshold algorithm.

$j \leftarrow 0$

// Initial Newton iteration uses full Jacobian

$\Delta\hat{\boldsymbol{v}}^{j+1} \leftarrow -\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{\boldsymbol{j}})^{-1}\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$

$\hat{\boldsymbol{v}}^{j+1} \leftarrow \hat{\boldsymbol{v}}^j + \Delta\hat{\boldsymbol{v}}^{j+1}$

// Main loop for Newton iterations

repeat

  $j \leftarrow j + 1$

  $\hat{\boldsymbol{J}}_h \leftarrow \text{Threshold}(\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j), h)$

  $\Delta\hat{\boldsymbol{v}}_h^{j+1} \leftarrow -\hat{\boldsymbol{J}}_h^{-1}\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$

  $\hat{\boldsymbol{v}}^{j+1} \leftarrow \hat{\boldsymbol{v}}^j + \Delta\hat{\boldsymbol{v}}_h^{j+1}$

  $\hat{\boldsymbol{\epsilon}} \leftarrow |\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) + \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}|$

  $\hat{\boldsymbol{t}} \leftarrow r_{tol}\text{minimum}\{|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)|, |\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)\Delta\hat{\boldsymbol{v}}_h^{j+1}|\} + a_{tol}\hat{\boldsymbol{u}}$

  $k \leftarrow \|\hat{\boldsymbol{\epsilon}} \oslash \hat{\boldsymbol{t}}\|_\infty$

  if $k < 0.1$ **then**

    $h \leftarrow h \times a$

  else if $k > 1$ **then**

    $h \leftarrow h/b$

  end if

  $\hat{\boldsymbol{n}} \leftarrow r_{tol}\min\{|\hat{\boldsymbol{v}}^j|, |\hat{\boldsymbol{v}}^{j+1}|\} + a_{tol}\hat{\boldsymbol{u}}$

until $\Delta\hat{\boldsymbol{v}}_{h,i}^{j+1} < \hat{\boldsymbol{n}}_i\,, \forall i \in \{1, 2, \ldots, NM\}$ **and**

  $\|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^{j+1})\|_\infty < a_{tol}$

The proposed method for adaptive Jacobian thresholding is given in Algorithm 4.1. In this algorithm,

a and b are the threshold increase and decrease factors, $\Delta\hat{\boldsymbol{v}}_{h,i}^{j+1}$ represents element i of $\Delta\hat{\boldsymbol{v}}_{h}^{j+1}$, and the ' $\oslash$ ' symbol denotes element-by-element division of two vectors (Hadamard division). The decision to raise or lower the threshold is controlled by element-wise division of $\hat{\boldsymbol{\epsilon}}$ by $\hat{\boldsymbol{t}}$ and then calculating:

$$k = \|\hat{\boldsymbol{\epsilon}} \oslash \hat{\boldsymbol{t}}\|_{\infty}. \tag{4.5}$$

For this thesis, if k is less than 0.1 then the threshold is increased and if k is greater than 1 then the threshold is decreased. The selection of this range will effect how often the threshold is changed. For example, in this thesis the range is $0.1 < k < 1$. If the lower range is decreased (for example, $0.00001 < k < 1$) then the threshold will not be increased as often as it would for the range used in this thesis. If the upper range is increased (for example, $0.1 < k < 100$), the threshold will be decreased less often than it would for the range used in this thesis. Increasing the upper range will cause more error to be allowed in the Newton updates before the threshold is lowered and could lead to an increase in number of iterations required to converge. Therefore, it is not recommended to set this to a high number. Setting the lower range to a lower number is safer but results in lower thresholds than could otherwise be utilized. The threshold increase and decrease factors are used to control how the threshold changes. In this study we have selected $a = 2$ and $b = 10$ so that the threshold will always decrease at a higher rate than it increases. Although a and b have to be selected by a designer when the simulation is run, they only affect the speed that the threshold is adjusted. Therefore, there is no risk of a user selected property leading to an over aggressive threshold provided reasonable values for a and b are selected.

4.2 Simulation Studies

Simulations were run on an HP Envy 17-J170ca using the Cardoon circuit simulator. The operating system is Ubuntu 18.04.3 LTS and the computer has 16GB DDR3 SDRAM and an 2.2GHz Intel Core i7-4702MQ Processor with Turbo Boost Technology (feature is enabled and can overclock the processors up to 3.2GHz). Simulations and thresholding were performed with the following parameters for each case study and mother wavelet: $a_{tol} = 10^{-7}$, $r_{tol} = 10^{-4}$, and an initial $h = 10^{-7}$. The UMFPack algorithm operates by performing a symbolic factorization step followed by a numeric factorization step. During the symbolic factorization step, sparse matrix column pre-ordering is performed to reduce fill-in[I] before the symbolic factorization is performed to estimate the structures of the L and U matrices. During each simulation in this study, the matrix column pre-ordering is calculated during the first iteration only. This pre-ordering is then re-used for each subsequent iteration. Thus, the sparse matrix pre-ordering is only performed once per simulation. The numerical factorization step performs the LU factorization of the Jacobian matrix to obtain the L and U factors.

[I]Fill-in refers to the number of elements in the L and U matrix that begin with values of zero and are changed to non-zero values by the end of factorization.

The results of simulations run with non-thresholded Jacobian matrices and adaptively thresholded Jacobian matrices (Algorithm 4.1) are presented in this section. Three types of derivative estimates were used for the time derivatives: The two point central differences method (D_{C2}), four point central differences method (D_{C4}), and the Fourier derivative method (Fourier). For more information on these methods, see Appendix A. Mother wavelets were selected from commonly used wavelet families to determine the effect of both the wavelet shape and number of vanishing moments on each of the circuits and, to maximize the sparsities of the vectors, the maximum level of transform allowed by each wavelet was used. The maximum level of transform is reached when the scaling function has been dilated to a scale where any further increase in scale would result in the scaling function being larger than the signal being transformed. For example, the Haar wavelet scaling function at level 1, once discretized, has two samples. Increasing the level of the transform will result in a scaling function with twice as many samples as the previous levels scaling function. Increasing the level again will result in 4 samples, then 8, then 16, and so on until the scaling function has 512 samples (this occurs at level 9 since $2^9 = 512$. Once level 9 is reached there will be 1 coefficient in the scaling subspace and 9 wavelet subspaces. The Db2 wavelet, once discretized, has 4 samples at level 1. The maximum level transform for this wavelet would be 7.[I] Since the scaling function is translated by a factor of 2 at each level of transform, the scaling and wavelet subspaces at each level will have half as many coefficients as the previous level. With the Db2 wavelet, this would result in a scaling subspace with $\frac{512}{2^7} = 4$ coefficients at the maximum possible level of transform. The number of scaling coefficients and wavelet subspaces for 512 time samples with the mother wavelets used in this study are shown in Table 4.1. The number of time samples was manually selected for this study and was determined experimentally as the minimum required number of samples for the nodal variable vectors to have no significant error in them with every test circuit (compared to simulations with a higher number of time samples).

Since the number of vanishing moments in the mother wavelet affects the maximum level of transform that is possible, the number of scaling coefficients is not the same for every wavelet. Wavelets with a higher number of vanishing moments will have a higher number of scaling coefficients. Since the scaling coefficients are not thresholded, wavelets with a higher number of vanishing moments will tend to lead to higher density vectors.

[I] Since $2^2 = 4$ and each increase in level increases the number of samples in the scaling function by a factor of 2, it would require 7 additional increases in level of transform for the scaling function to have 512 samples.

Table 4.1: Number of scaling coefficients and wavelet subspaces for 512 time samples with selected mother wavelets.

Wavelet	Number of scaling coefficients	Number of wavelet subspaces	Wavelet	Number of scaling coefficients	Number of wavelet subspaces
Haar	1	9	Coif12	128	2
Db2	4	7	Coif16	128	2
Db3	8	6	Sym2	4	7
Db4	8	6	Sym3	8	6
Db8	16	5	Sym4	8	6
Db12	32	4	Sym8	16	5
Db16	32	4	Sym12	32	4
Dmey	64	3	Sym16	32	4
Coif2	16	5	Bior1.1	1	9
Coif3	32	4	Bior4.4	16	5
Coif4	32	4	Bior6.8	32	4
Coif8	64	3			

Presented in this section are:

1. Input and output nodal waveforms.
2. Number of iterations.
3. Average $\hat{\boldsymbol{v}}$ density.
4. Average Jacobian density.
5. Variation of h per iteration.
6. Simulation times for SSTD and SSWA (with no thresholding, with adaptive thresholding, and with a static threshold) with some selected wavelets using the D_{C2} derivative.
7. Change in average number of non-zero elements in the L and U matrices.

The input and output nodal waveforms are presented to show that there is no difference between the final result of the thresholded and non-thresholded simulations. The number of iterations is presented to show that there is no large change in number of iterations, if any, when applying the Jacobian thresholding. The average density of $\hat{\boldsymbol{v}}$ for all iterations with the different mother wavelets is shown for each simulation. This is important to estimate the potential savings that could be obtained by methods that take advantage of low density nodal vectors [10].

Many of the wavelet coefficients in the nodal variables will have negligible amplitudes. For this reason, the nodal waveforms are thresholded to remove the low amplitude coefficients. Since each nodal

variable has its own waveform, the wavelet coefficients for each node are thresholded independently with a threshold that is selected using the following formula:

$$t_v = r_{tol}\|\hat{\boldsymbol{v}}_k\|_\infty + a_{tol}; \tag{4.6}$$

where $\hat{\boldsymbol{v}}_k$ is a vector of wavelet domain nodal voltages for node k. All wavelet coefficients for node k that have an amplitude that is below t_v are then set to zero (the scaling coefficients are not thresholded). The average Jacobian density and variation of h are presented to show how much h affects Jacobian sparsity.

To illustrate the potential trade off between computational savings and overhead of the thresholding method, the simulation time for SSTD is also presented and compared with the simulation times of the wavelets that provided the sparsest Jacobian matrices and $\hat{\boldsymbol{v}}$ vectors with the D_{C2} derivative. The Haar wavelet provided the lowest average Jacobian densities for all circuits except for the ultrasound transducer driver with Jacobian matrix thresholding where the Db4 wavelet provided the lowest average Jacobian density. The sparsest wavelet for the inverter chain circuit was the Sym3 wavelet. The Db2 and Sym2 wavelets were tied for the Gilbert cell, ultrasound transducer driver, and transmission line circuits. The Db4 wavelet provided the sparsest $\hat{\boldsymbol{v}}$ vectors for the stepped impedance filter circuit. When applying the adaptive threshold, the Jacobian matrix pre-ordering from the first iteration is re-used for every subsequent iteration.[I] This could potentially affect the fill-ins that occur when factorizing the thresholded Jacobian matrices at each subsequent iteration. For this reason, the change in average number of non-zero elements in the L and U matrices between the thresholded and un-thresholded simulations is compared. Since there can be a small effect on matrix and vector densities when applying the adaptive thresholding method to SSTD, the number of iterations, $\hat{\boldsymbol{v}}$ densities, Jacobian matrix densities, simulation times, numerical factorization times, symbolic factorization times, and difference in L and U matrix densities are included for thresholded SSTD for comparison.

4.2.1 Inverter Chain

The first case study is the CMOS inverter chain circuit shown in Figure 4.1. The ALD1103 IC used in these simulations is made up of 2 matched pairs of MOSFETs. They were modelled using the intrinsic portion of the EKV 2.6 model [56].[II] The model parameters were extracted from measurements from an ALD1103 chip. There are 7 nodal variables with 512 time samples for this circuit which results in 3584 unknowns. The non-thresholded (solid line) and thresholded (dashed line) input and output voltage waveforms are shown in Figure 4.2. The vector thresholding method has an insignificant effect on the time domain representation of the signals.

[I] The first iteration Jacobian matrix is not thresholded and, therefore, the thresholding method will not affect the calculated pre-ordering.

[II] For more information see the ekv_i model at: https://vision.lakeheadu.ca/cardoon/device_library.html#ekv-i-intrinsic-epfl-ekv-2-6-mosfet

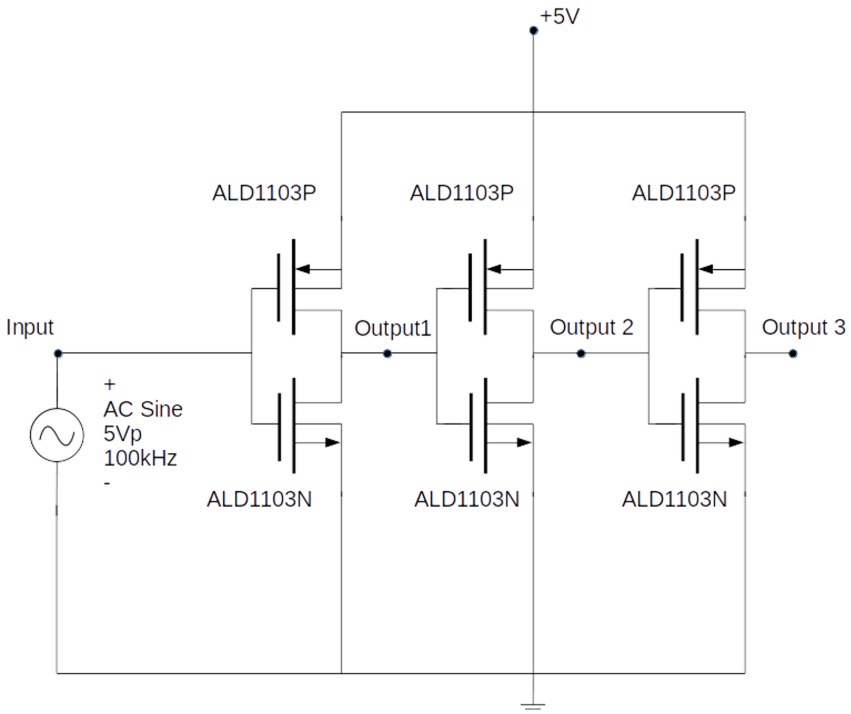

Figure 4.1: Inverter chain schematic.

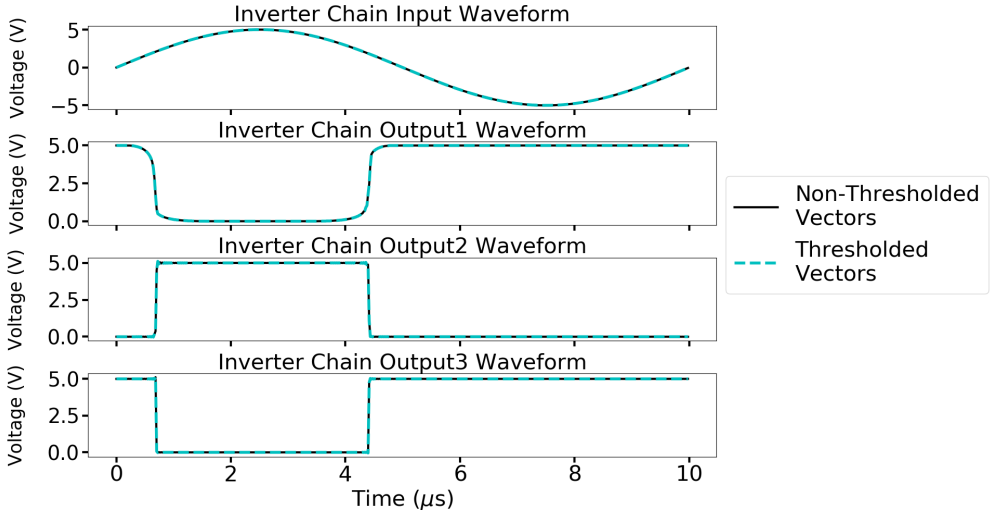

Figure 4.2: Input and output waveforms for the inverter chain. Dashed line represents the signals after they are thresholded.

Figure 4.3 shows the Jacobian density for all simulations and Table 4.2 shows a detail for the lowest un-thresholded and thresholded densities, with the thresholded densities denoted between parentheses. The time domain simulation provided the lowest Jacobian densities for D_{C2} and D_{C4}.

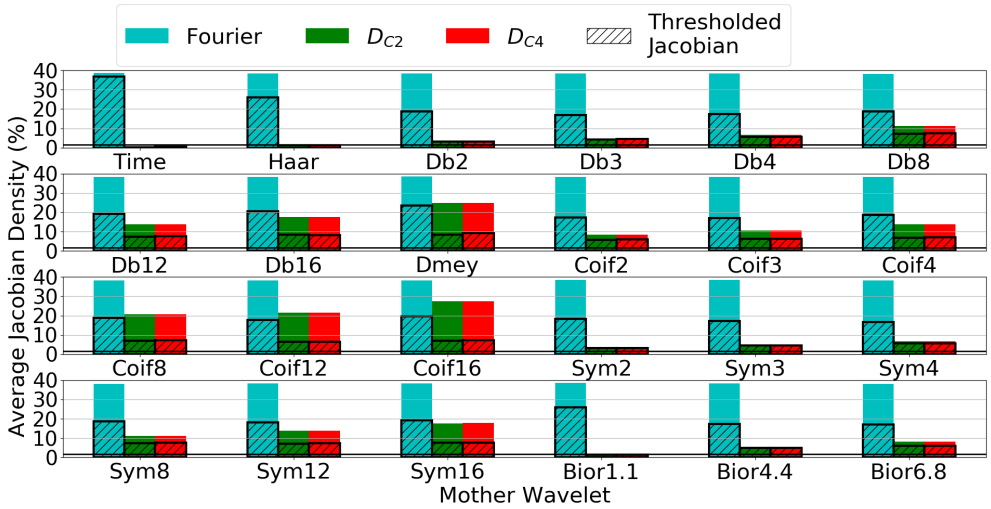

Figure 4.3: Average Jacobian densities for the inverter chain.

Table 4.2: Inverter chain lowest Jacobian densities in %. Densities for the adaptive Jacobian threshold simulations are shown in parentheses.

Domain	time	Haar	Bior1.1
D_{C2} (thresholded)	0.22 (0.21)	1.62 (1.36)	1.59 (1.36)
D_{C4} (thresholded)	0.37 (0.35)	1.69 (1.33)	1.69 (1.33)

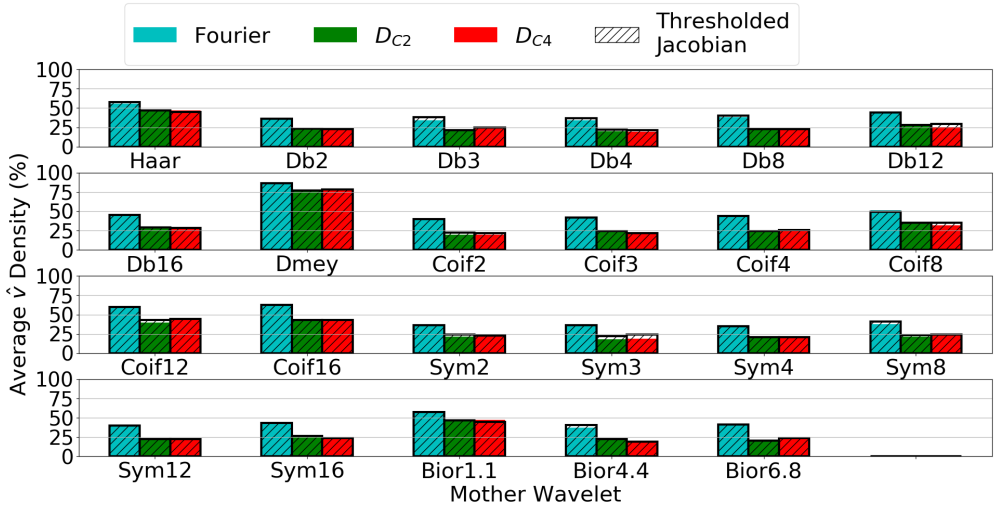

Figure 4.4: Average $\hat{v}$ densities for the inverter chain.

For wavelet simulations there are no significant density differences between the D_{C2} and D_{C4} derivative methods. The Haar wavelet and the Biorthogonal 1.1 wavelet, which is an Biorthogonal equivalent to the Haar wavelet, performed best in terms of Jacobian densities. As expected, the Fourier derivative produces the highest Jacobian densities and it is clear that this kind of derivative approximation can not compete

with finite differences.

The average density of $\hat{\boldsymbol{v}}$ with the non-thresholded and thresholded Jacobian matrices, is shown in Figure 4.4. The Daubechies and Symlet wavelets have similar densities and there is a slight increase in density as the number of vanishing moments increases. The Haar, Bior1.1, Coif12, Coif16, and Dmey wavelets had the highest densities. This is due to how the wavelets represent the nodal waveforms. The smooth waveforms of the Input and Output1 nodes, along with any smooth waveforms for nodes that are not shown, will not be sparsely represented by the Haar wavelet and cannot be thresholded much before a large amount of time domain error is introduced. Similarly, the Output2 and Output3 nodal waveforms, along with any sharply changing waveforms that are not shown, will not be represented sparsely by the Dmey and Coiflet wavelets. The other wavelets represent a middle ground between the aforementioned wavelets and are sparser as a result. For some simulations, the effect of thresholding the Jacobian matrix results in a small variation of the density of $\hat{\boldsymbol{v}}$ at some iterations. The error introduced by using $\hat{\boldsymbol{v}}_h^{j+1}$ can change the waveforms enough to alter the density of $\hat{\boldsymbol{v}}$ and a small change in density can be expected with the adaptive thresholding method.

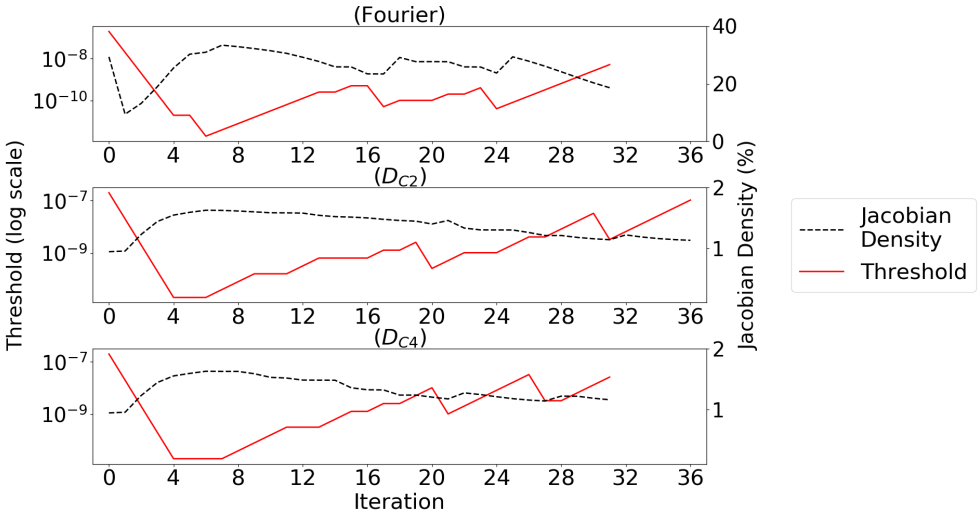

Figure 4.5: Variation of h and Jacobian density at each iteration for the inverter chain using the Haar wavelet.

The variation of h and the Jacobian density using the Haar wavelet, which led to the lowest average Jacobian matrix density when thresholding was applied, at each iteration are shown in Figure 4.5. The threshold value has relatively large variations that, with the exception of the Fourier derivatives, do not have a large effect on Jacobian density.

The number of iterations necessary for convergence is shown in Figure 4.6. The difference in the number of iterations seen with some wavelet/derivative combinations is due to the error introduced by

using $\Delta\hat{\boldsymbol{v}}_h^{j+1}$ instead of $\Delta\hat{\boldsymbol{v}}^{j+1}$ when the thresholded Jacobian matrix is used. The difference represents an extra error that has to be accounted for and can lead to extra iterations. It is also possible for this error to be a 'lucky guess' and cause the number of iterations to be lower. We can see from Figure 4.5 that a single threshold does not perform optimally, in terms of error, for all iterations and h must be adapted as the simulation runs to keep the error at a minimum. The thresholding method is effectively adapting this threshold and keeping the error low enough to only cause a small alteration in the number of iterations required for the simulation to converge for most cases.

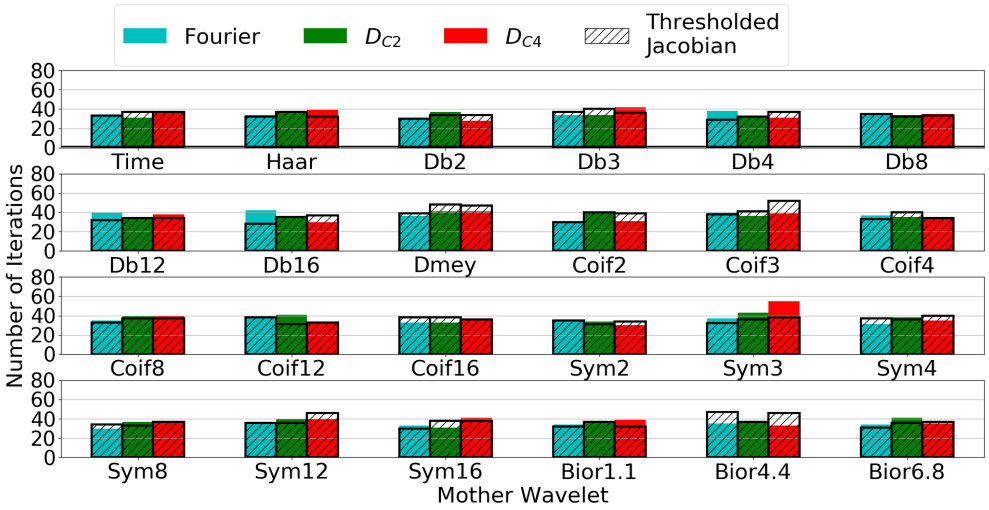

Figure 4.6: Number of iterations for the inverter chain.

There are some mother wavelets, even when no Jacobian thresholding is used, that required a different number iterations to converge on a solution than the time domain simulation. This is likely due to rounding errors introduced into the vectors during the forward and backward wavelet transforms. Given the observed disadvantages of using the Fourier derivative in the considered simulations (*i.e.* when the error function is not defined in the frequency domain), this derivative is not included for the remaining examples.

The simulation run times for the Inverter chain for the time domain and Haar and Sym2 wavelets are shown in Table 4.3. The static threshold selected for this circuit was 10^{-9} and is, roughly, the average threshold used by the adaptive thresholding method (see Figure 4.5). All wavelets exhibit a speedup when adaptive Jacobian matrix thresholding is applied and, due to a reduction in number of iterations, the speedup in one case was enough for the adaptively thresholded simulation to be faster than SSTD. Based on the per-iteration times, the adaptively thresholded simulations would still exhibit a speedup over un-thresholded SSWA if the number of iterations was unchanged. The static threshold tended to be slower than the adaptive threshold method. The static threshold matched the per-iteration numeric

factorization time of the adaptive thresholding method with the Haar simulation with the D_{C4} derivative and exhibited lower per-iteration numerical factorization times than the adaptive thresholding method with both Sym3 simulations.

Table 4.3: Total simulation, symbolic factorization, and numeric factorization timing data, in seconds, for the inverter chain circuit with the Haar and Sym3 wavelets. The per-iteration times are shown in brackets. The static threshold value was 10^{-9}. Green font indicates the cases where the adaptive thresholding method is faster than the non-thresholded method while blue font indicates the cases where the static thresholding method is faster than the adaptive thresholding method.

		D_{C2}			D_{C4}		
		No Thresh	Adaptive	Static	No Thresh	Adaptive	Static
Time	Total	30.66 [0.989]	36.04 [0.974]	39.23 [1.060]	35.83 [0.995]	36.27 [0.980]	35.41 [1.041]
	Symbolic	0.162 [0.005]	0.167 [0.005]	0.169 [0.005]	0.175 [0.005]	0.179 [0.005]	0.177 [0.005]
	Numeric	0.437 [0.014]	0.458 [0.012]	0.456 [0.012]	0.463 [0.013]	0.471 [0.013]	0.476 [0.014]
	Iterations	31	37	37	36	37	34
Haar	Total	43.35 [1.204]	41.40 [1.118]	41.79 [1.161]	43.82 [1.123]	35.53 [1.110]	38.33 [1.128]
	Symbolic	0.295 [0.008]	0.276 [0.008]	0.291 [0.008]	0.281 [0.007]	0.261 [0.008]	0.274 [0.008]
	Numeric	0.926 [0.026]	0.852 [0.023]	0.898 [0.025]	0.982 [0.025]	0.780 [0.024]	0.831 [0.024]
	Iterations	36	37	36	39	32	34
Sym3	Total	60.98 [1.418]	49.70 [1.381]	40.92 [1.240]	85.30 [1.551]	51.30 [1.350]	47.07 [1.272]
	Symbolic	0.528 [0.012]	0.460 [0.013]	0.362 [0.011]	0.730 [0.013]	0.470 [0.0124]	0.391 [0.011]
	Numeric	3.920 [0.091]	3.110 [0.086]	1.675 [0.051]	5.670 [0.103]	3.217 [0.085]	1.808 [0.049]
	Iterations	43	36	33	55	38	37

4.2.2 Gilbert Cell

The second case study is the Gilbert cell circuit shown in Figure 4.7. There are 15 nodal variables in this circuit with 512 time samples which results in 7680 variables. The non-thresholded (solid line) and thresholded (dashed line) input and output voltage waveforms are shown in Figure 4.8. This circuit has waveforms with shapes that are a mixture of sharply changing pulses and smoothly changing curves.

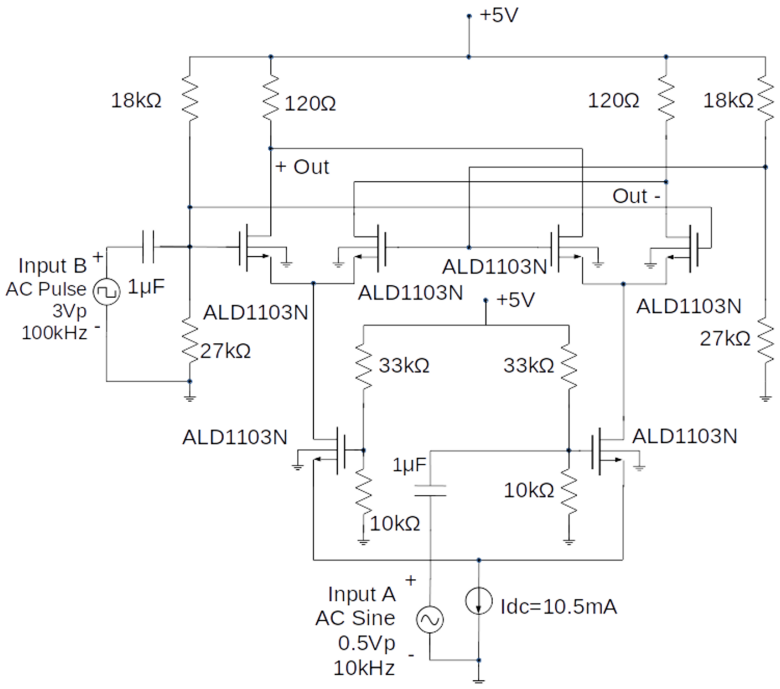

Figure 4.7: Gilbert cell schematic.

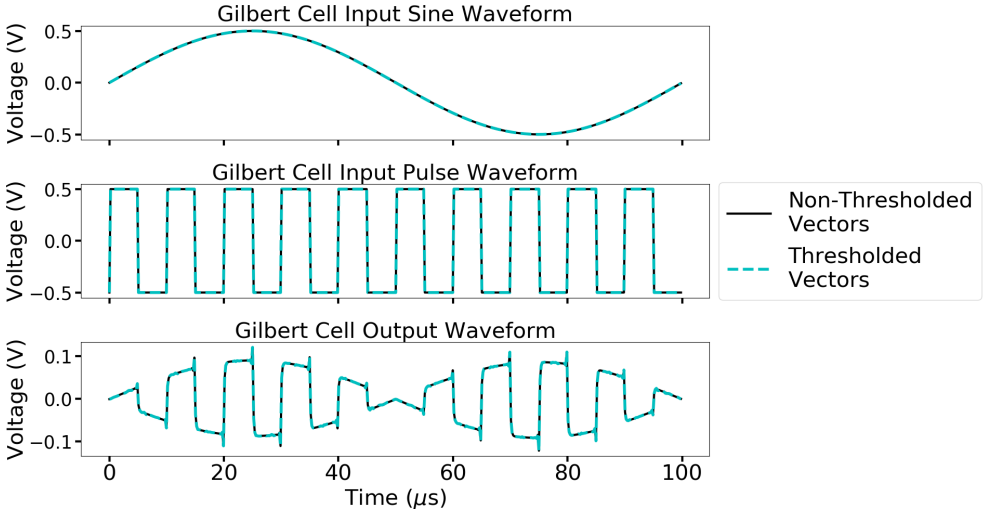

Figure 4.8: Input and output waveforms for the Gilbert cell. Dashed line represents the signals after they are thresholded.

The Jacobian densities are given in Figure 4.9 and Table 4.4. The wavelets that have the lowest number of vanishing moments lead to the sparsest Jacobian matrices and the greatest reduction in density, when thresholding is applied, occurred with wavelets with the highest number of vanishing moments. The time domain led to the sparsest Jacobian matrices. The variation of h and the Jacobian density using the Haar wavelet at each iteration is shown in Figure 4.10. Only small variations in the density occur as the

Jacobian matrix threshold varies.

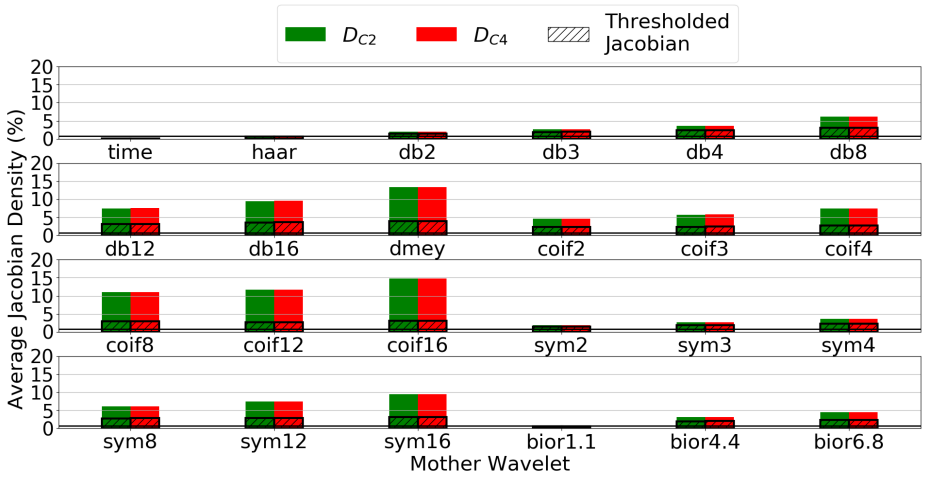

Figure 4.9: Average Jacobian densities for the Gilbert cell.

Table 4.4: Gilbert cell lowest Jacobian densities in %. Densities for the adaptive Jacobian threshold simulations are shown in parentheses.

Domain	time	Haar	Bior1.1
D_{C2} (thresholded)	0.13 (0.11)	0.78 (0.58)	0.78 (0.58)
D_{C4} (thresholded)	0.21 (0.17)	0.87 (0.59)	0.87 (0.59)

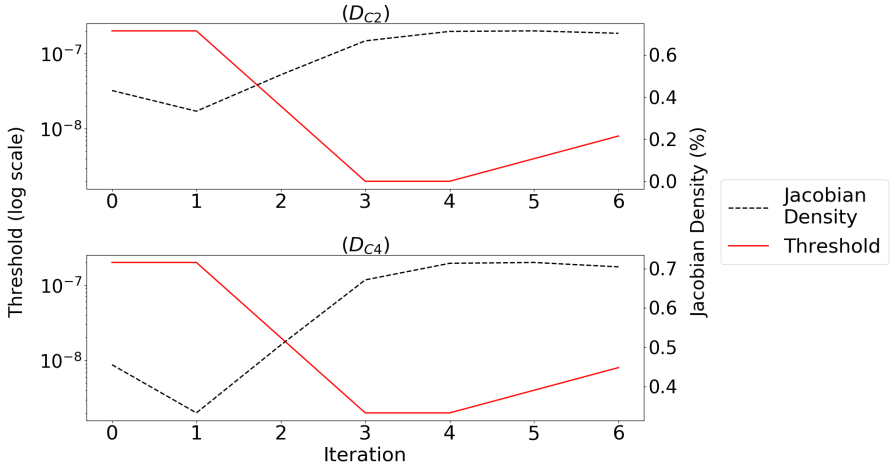

Figure 4.10: Variation of h and Jacobian density at each iteration for the Gilbert cell using the Haar wavelet.

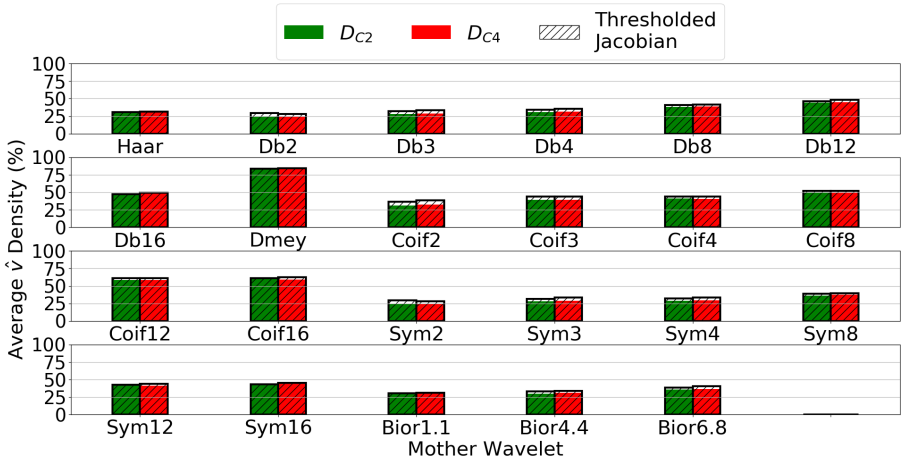

Figure 4.11: Average $\hat{\boldsymbol{v}}$ densities for the Gilbert cell.

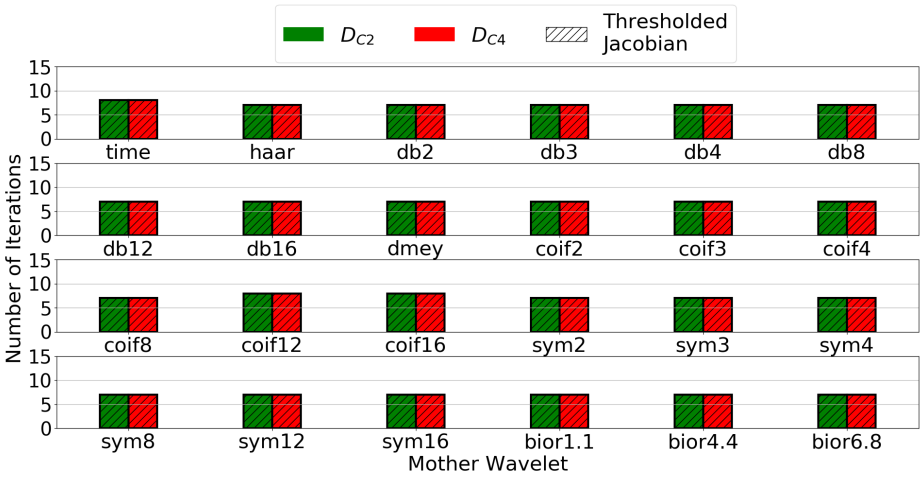

Figure 4.12: Number of iterations for the Gilbert cell.

The average density of $\hat{\boldsymbol{v}}$ with and without adaptive Jacobian thresholding is shown in Figure 4.11. The lowest density representations of $\hat{\boldsymbol{v}}$ occur with the wavelets with the lowest number of vanishing moments. The number of iterations for each of the simulations is shown in Figure 4.12. The time domain simulations required 8 iterations to converge while the wavelet domain simulations required 7 and the number of iterations remains unchanged when Jacobian thresholding is used which indicates that the thresholding algorithm successfully keeps the error introduced by Jacobian thresholding at a minimum with this test case.

The simulation run times for the Gilbert cell for the time domain and Haar, Db2, and Sym2 wavelets are shown in Table 4.5. The static threshold used for this circuit was 10^{-8} which represents a rough average of the threshold values used by the adaptive thresholding method (see Figure 4.10). All the adaptive threshold simulations exhibited a small speedup over the non-thresholded case and the per-

iteration numeric factorization times are all lower with the adaptively thresholded simulations. None of the wavelet simulations were faster than SSTD. The adaptive thresholding method exhibited lower per-iteration numeric factorization times than the static threshold for all but the Haar wavelet simulations and the time domain simulation with the D_{C4} derivative.

Table 4.5: Total simulation, symbolic factorization, and numeric factorization timing data, in seconds, for the Gilbert cell circuit with the Haar, Db2, and Sym2 wavelets. The per-iteration times are shown in brackets. The static threshold value was 10^{-8}. Green font indicates the cases where the adaptive thresholding method is faster than the non-thresholded method while blue font indicates the cases where the static thresholding method is faster than the adaptive thresholding method.

		D_{C2}			D_{C4}		
		No Thresh	Adaptive	Static	No Thresh	Adaptive	Static
Time	Total	13.50 [1.687]	14.25 [1.782]	16.62 [2.078]	13.84 [1.730]	14.61 [1.826]	16.18 [2.023]
	Symbolic	0.6799 [0.085]	0.6955 [0.087]	0.7287 [0.091]	0.6855 [0.086]	0.6989 [0.087]	0.793 [0.099]
	Numeric	3.839 [0.480]	4.051 [0.506]	4.189 [0.524]	3.903 [0.488]	4.036 [0.505]	3.919 [0.490]
	Iterations	8	8	8	8	8	8
Haar	Total	17.60 [2.514]	16.49 [2.356]	16.52 [2.360]	17.40 [2.485]	16.67 [2.381]	17.01 [2.431]
	Symbolic	0.797 [0.114]	0.766 [0.109]	0.769 [0.110]	0.777 [0.111]	0.784 [0.112]	0.762 [0.109]
	Numeric	5.021 [0.717]	4.565 [0.652]	4.514 [0.645]	4.892 [0.699]	4.519 [0.646]	4.346 [0.621]
	Iterations	7	7	7	7	7	7
Db2	Total	25.06 [3.581]	23.02 [3.288]	24.67 [3.525]	25.31 [3.616]	23.61 [3.372]	25.00 [3.572]
	Symbolic	0.870 [0.124]	0.949 [0.136]	0.944 [0.135]	0.870 [0.124]	0.969 [0.138]	0.930 [0.133]
	Numeric	9.690 [1.384]	8.426 [1.204]	9.258 [1.323]	9.855 [1.408]	8.113 [1.159]	9.504 [1.358]
	Iterations	7	7	7	7	7	7
Sym2	Total	24.744 [3.535]	22.902 [3.272]	23.909 [3.416]	25.242 [3.606]	22.764 [3.252]	24.909 [3.559]
	Symbolic	0.853 [0.122]	0.960 [0.137]	0.928 [0.133]	0.855 [0.122]	0.961 [0.137]	0.936 [0.134]
	Numeric	9.834 [1.405]	8.3794 [1.197]	9.074 [1.296]	9.733 [1.390]	7.925 [1.132]	9.507 [1.358]
	Iterations	7	7	7	7	7	7

4.2.3 Ultrasound Transducer Driver

The third case study is the ultrasound transducer driver circuit, shown in Figure 4.13, which is similar to the one discussed in [57]. There are 8 nodal variables in this circuit with 512 time samples which

results in 4096 unknowns. The transducer in this circuit is modelled in the frequency domain, using the measured S-parameters in a frequency range from 800kHz to 10MHz. Outside of this range, the transducer is modelled as a capacitor.

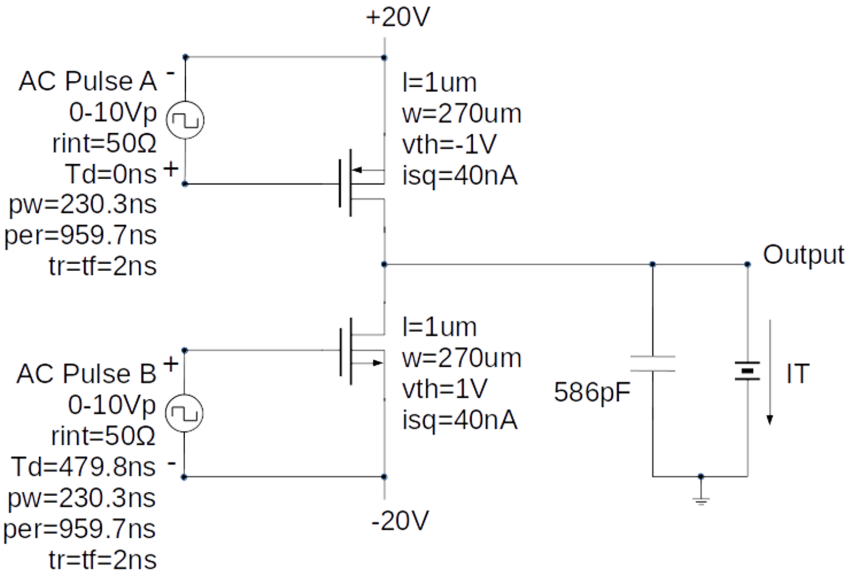

Figure 4.13: Ultrasound transducer driver schematic.

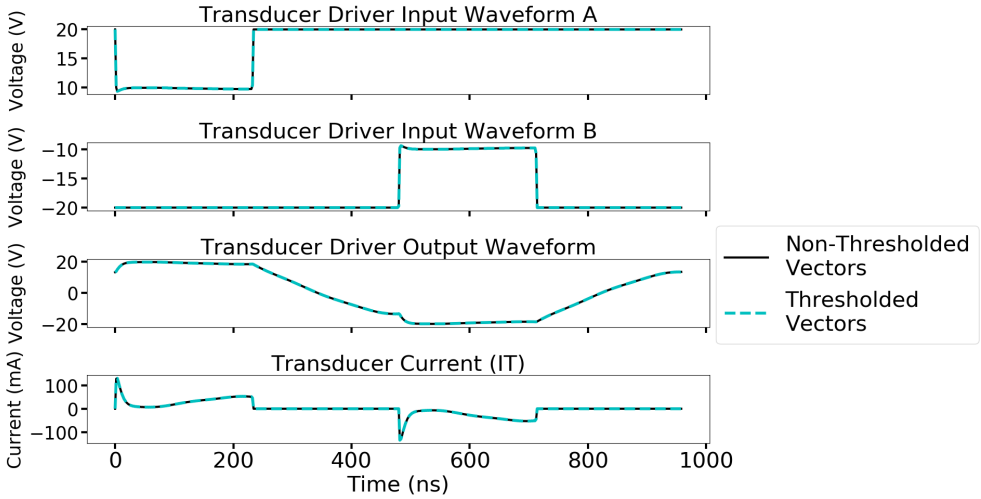

Figure 4.14: Input and output waveforms for the ultrasound transducer driver. Dashed line represents the signals after they are thresholded.

The MOSFETs are modelled using the ACM model [58].[I] Figure 4.14 shows the input, output, and transducer current waveforms. The vector thresholding method causes no significant change in the time

[I]For more information see the acms_i model at: https://vision.lakeheadu.ca/cardoon/device_library.html#acms-i-simplified-acm-intrinsic-mosfet

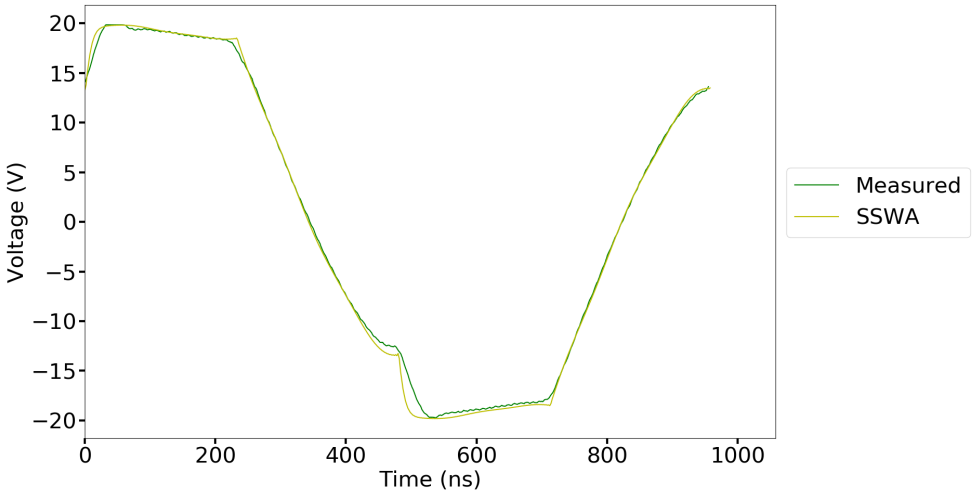

Figure 4.15: Measured vs Simulated output waveform of the ultrasound transducer driver.

domain representation of the nodal waveforms. Figure 4.15 shows the simulated SSWA output waveform vs data measured from a constructed circuit (data from [57]). The simulated results closely resemble the measured output.

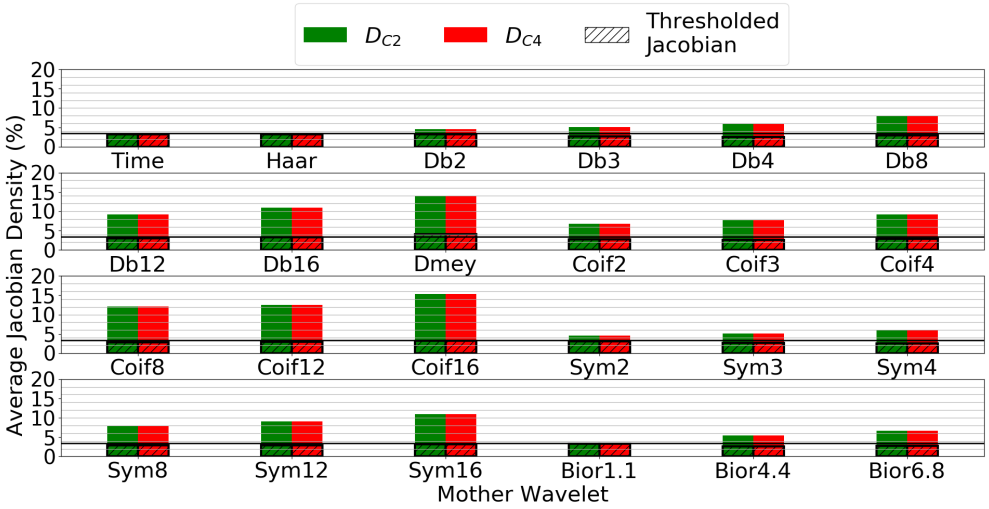

Figure 4.16: Average Jacobian densities for the ultrasound transducer driver.

The Jacobian density for each simulation is given in Figure 4.16. Densities tend to be higher with some wavelets due to the presence of the frequency-domain transducer model. As seen in Equation (2.14), the dense blocks contributed by that model are independent of the method used to approximate time derivatives.

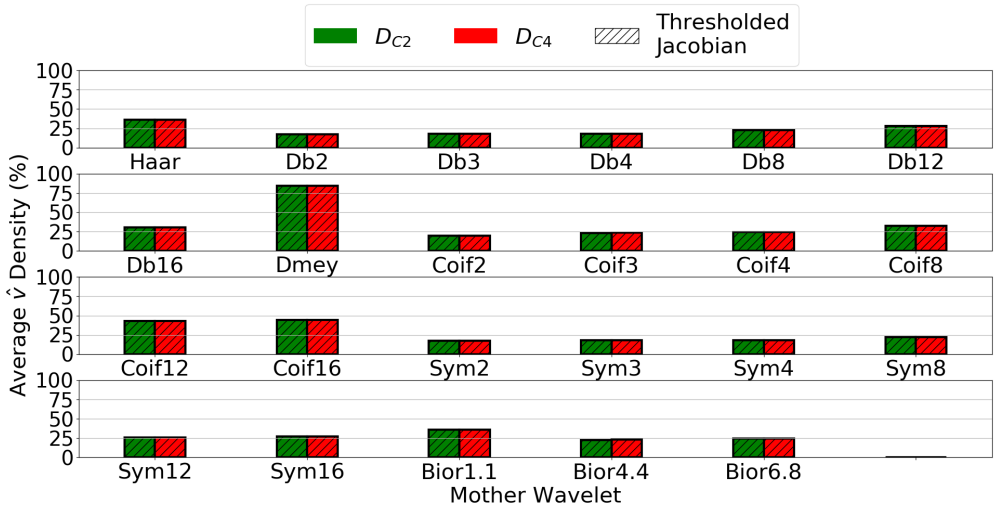

Figure 4.17: Average $\hat{\boldsymbol{v}}$ densities for the ultrasound transducer driver.

The wavelets with the lowest number of vanishing moments lead to the sparsest Jacobian matrices while the wavelets with the highest number of vanishing moments lead to the greatest decrease in Jacobian density when thresholding is applied. The time domain leads to the sparsest Jacobian matrices when no thresholding is applied but, with thresholding, some wavelets lead to slightly sparser Jacobian matrices than the time domain.

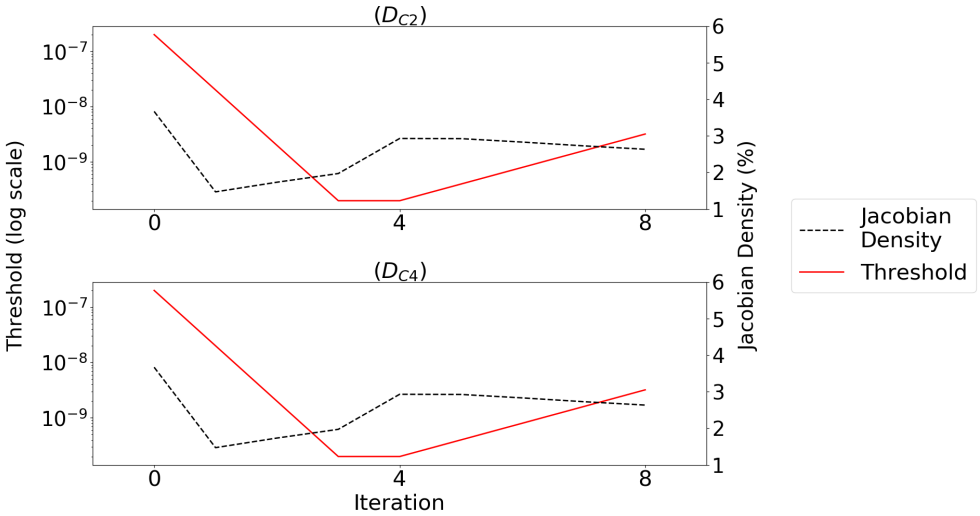

Figure 4.18: Variation of h and Jacobian density at each iteration for the ultrasound transducer driver using the Db4 wavelet.

Figure 4.17 shows the density of $\hat{\boldsymbol{v}}$. The wavelets with the lowest number of vanishing moments tend to lead to the lowest density representations of $\hat{\boldsymbol{v}}$ with the exception of the Haar wavelet which, due to the presence of very smooth waveforms, leads to higher density representations of $\hat{\boldsymbol{v}}$ than other wavelets with a similar number of vanishing moments. The Db4 wavelet leads to the lowest average $\hat{\boldsymbol{v}}$ vector densities.

This suggests the Db4 wavelet has a higher correlation to $\hat{\boldsymbol{v}}$ on average than the other wavelets.

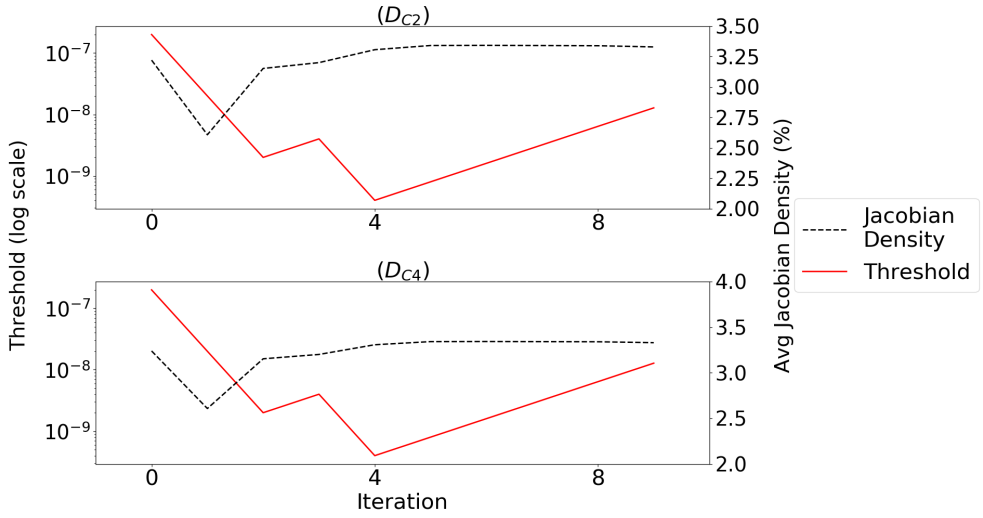

Figure 4.19: Variation of h and Jacobian density at each iteration for the ultrasound transducer driver using the Haar wavelet.

The variation of h and the Jacobian density are shown in Figure 4.18 for the Db4 wavelet, which led to the sparsest Jacobian matrices when thresholding was applied. There are no large variations in Jacobian density observed as the threshold changes. The variation of h and the Jacobian density with the Haar wavelet are shown in Figure 4.19. There are no large variations in Jacobian density observed as the threshold changes. The range of values h took during the simulation is roughly similar to the range of values of h seen with the Db4 wavelet but the effect of thresholding on the Db4 Jacobian density is more significant.

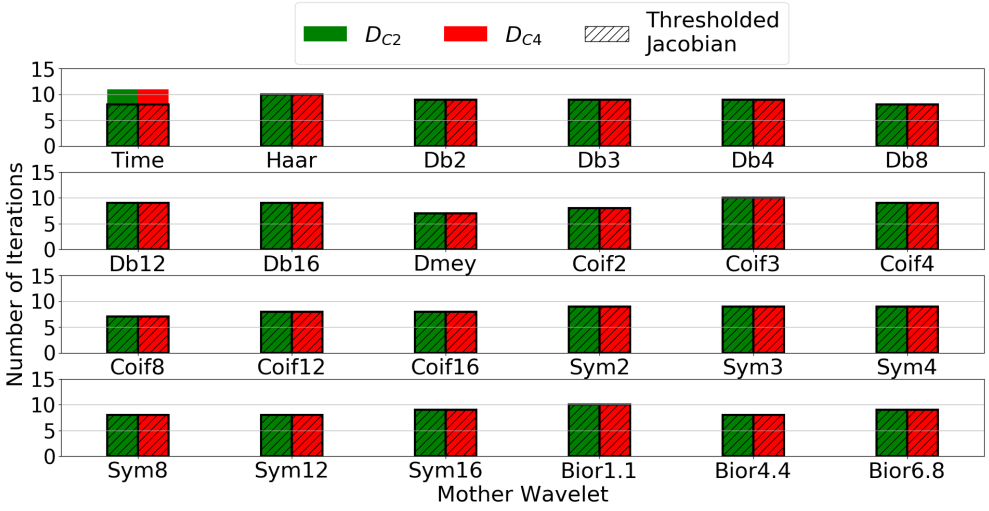

Figure 4.20: Number of iterations for the ultrasound transducer driver.

Table 4.6: Total simulation, symbolic factorization, and numeric factorization timing data, in seconds, for the ultrasound transducer driver circuit with the Haar, Db2, Db4, and Sym2 wavelets. The per-iteration times are shown in brackets. The static threshold value was 5×10^{-9}. Green font indicates the cases where the adaptive thresholding method is faster than the non-thresholded method while blue font indicates the cases where the static thresholding method is faster than the adaptive thresholding method.

		D_{C2}			D_{C4}		
		No Thresh	Adaptive	Static	No Thresh	Adaptive	Static
Time	Total	11.71 [1.065]	20.65 [2.581]	27.20 [2.472]	35.07 [3.188]	26.78 [3.347]	35.90 [3.264]
	Symbolic	0.305 [0.028]	0.273 [0.034]	0.361 [0.033]	0.296 [0.027]	0.273 [0.034]	0.337 [0.031]
	Numeric	6.703 [0.609]	16.67 [2.083]	21.79 [1.981]	30.06 [2.733]	22.75 [2.844]	30.29 [2.753]
	Iterations	11	8	11	11	8	11
Haar	Total	21.38 [2.138]	18.51 [1.851]	19.56 [1.956]	22.10 [2.210]	20.19 [2.019]	20.74 [2.074]
	Symbolic	0.329 [0.033]	0.297 [0.030]	0.319 [0.032]	0.342 [0.034]	0.296 [0.030]	0.301 [0.030]
	Numeric	16.04 [1.604]	13.19 [1.319]	14.16 [1.416]	16.53 [1.653]	14.77 [1.477]	15.38 [1.538]
	Iterations	10	10	10	10	10	10
Db2	Total	17.59 [1.955]	12.54 [1.393]	13.66 [1.518]	17.16 [1.907]	12.33 [1.370]	13.43 [1.492]
	Symbolic	0.415 [0.046]	0.271 [0.030]	0.289 [0.032]	0.398 [0.044]	0.271 [0.030]	0.282 [0.031]
	Numeric	11.56 [1.284]	7.103 [0.789]	8.194 [0.910]	11.27 [1.252]	6.915 [0.768]	8.109 [0.901]
	Iterations	9	9	9	9	9	9
Db4	Total	24.56 [2.729]	12.71 [1.412]	13.83 [1.537]	23.58 [2.620]	12.91 [1.435]	13.86 [1.541]
	Symbolic	0.530 [0.059]	0.270 [0.030]	0.285 [0.032]	0.520 [0.058]	0.278 [0.031]	0.298 [0.033]
	Numeric	17.66 [1.962]	7.070 [0.786]	7.908 [0.879]	16.630 [1.848]	7.200 [0.800]	7.940 [0.882]
	Iterations	9	9	9	9	9	9
Sym2	Total	16.76 [1.862]	12.25 [1.361]	14.27 [1.586]	17.04 [1.893]	12.15 [1.350]	13.37 [1.485]
	Symbolic	0.406 [0.045]	0.270 [0.030]	0.299 [0.033]	0.390 [0.043]	0.273 [0.030]	0.281 [8.112]
	Numeric	10.89 [1.211]	6.929 [0.770]	8.712 [0.968]	11.11 [1.235]	6.783 [0.754]	8.112 [0.901]
	Iterations	9	9	9	9	9	9

The number of iterations for each of the simulations is shown in Figure 4.20. Thresholding caused no change in the number of iterations required for convergence for the wavelets but did alter the number of iterations for the time domain. Since the number of iterations remains unchanged when Jacobian

thresholding is applied with the wavelet domain simulations, the error is kept low enough to not increase the number of iterations necessary to achieve convergence when Jacobian thresholding is applied. The time domain simulations have less coefficients in the Jacobian matrix that can be removed via thresholding without introducing a large amount of error. The error in the time domain simulations with thresholding, in this case, was a 'lucky guess' and led to a reduction in number of iterations.

The simulation run times for the transducer driver for the time domain and Haar, Db4, and Sym2 wavelets are shown in Table 4.6. The static threshold used for this circuit was 5×10^{-9} which represents a rough average of the threshold values used by the adaptive thresholding method (see Figures 4.18 and 4.19). There are significant speedups with all but one of the wavelets when adaptive Jacobian matrix thresholding is applied. The average Jacobian matrix density is not significantly reduced by adaptive Jacobian matrix thresholding with the Haar wavelet and, therefore, the speedup is also small. The Db4 wavelet did not lead to the lowest numeric factorization times even though the Jacobian matrix density was lower than the Sym2 wavelet when adaptive thresholding was applied. Additionally, there is no significant reduction in Jacobian density with the Haar wavelet but a speedup is still observed. All of the non-thresholded and adaptively thresholded simulations with the D_{C4} derivative ran faster than the time domain. The adaptive threshold led to lower per-iteration numeric factorization times than the static threshold for all but the time domain simulations. For the Db2 and Sym2, the per-iteration numeric factorization times are significantly higher with the static threshold than the adaptive thresholding method.

4.2.4 Transmission Line Circuit

The fourth case study is the transmission line circuit shown in Figure 4.21, which is based on an example circuit in [59] with an added inverter stage at the output. There are 18 nodal variables in this circuit with 512 time samples which results in 9216 variables. The transmission line is modelled in the frequency domain by its scattering parameters, which are generated using its physical characteristics: length of 1m, effective relative dielectric constant of 21, attenuation of 1 dB/m, and a characteristic impedance of 50 Ω. The MOSFETs for the inverter stage use the intrinsic portion of the EKV 2.6 model [56][I] with 0.5μm CMOS parameters (EKV v2.6 parameters for 0.5μm CMOS EPFL-EKV, 1999)[II] and have a channel length of 0.5μm and channel width of 10μm. The non-thresholded (solid line) and thresholded (dashed line) input and output waveforms and the voltages at two of the transmission line ports are shown in Figure 4.22. The vector thresholding method causes no significant change in the time domain representation of the nodal waveforms.

[I] For more information see the ekv_i model at: https://vision.lakeheadu.ca/cardoon/device_library.html#ekv-i-intrinsic-epfl-ekv-2-6-mosfet

[II] https://www.epfl.ch/labs/iclab/wp-content/uploads/2019/02/ekv_v262.pdf

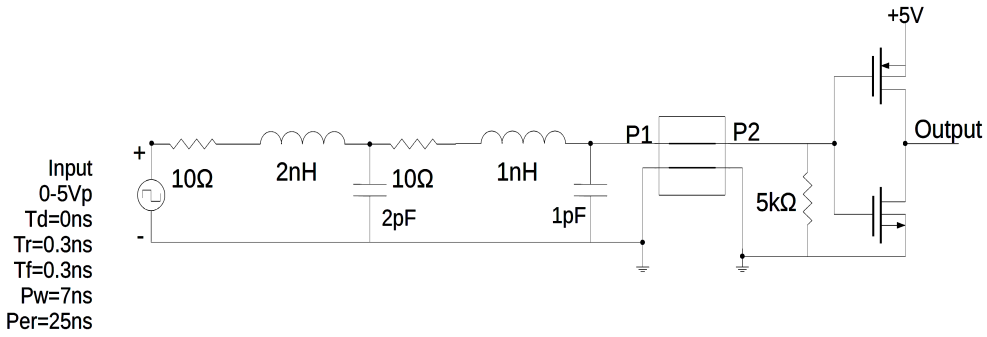

Figure 4.21: Transmission line circuit schematic.

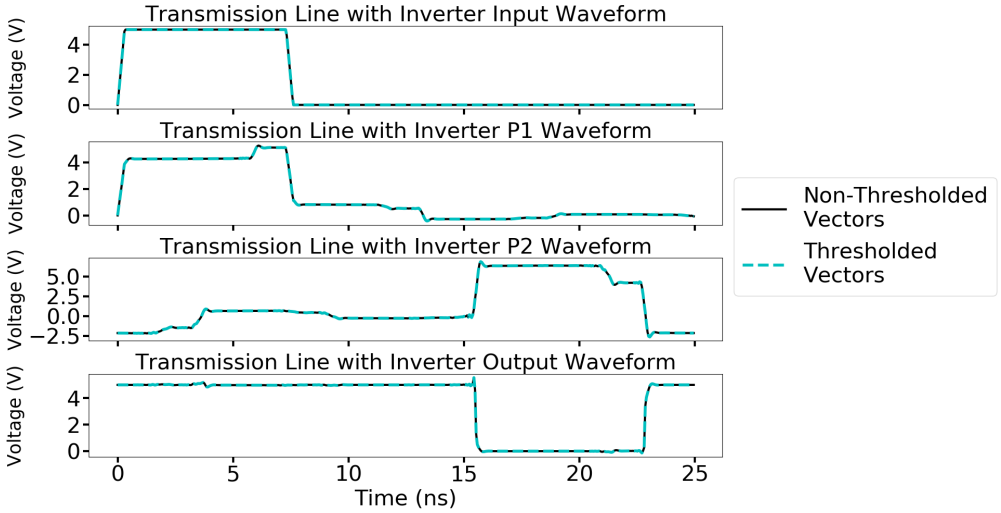

Figure 4.22: Input and output waveforms for the transmission line circuit. Dashed line represents the signals after they are thresholded.

The Jacobian densities are given in Figure 4.23. The wavelets with the lowest number of vanishing moments provide the lowest density Jacobian matrices while the greatest reduction in density occurs with wavelets that have the highest number of vanishing moments. When Jacobian thresholding is applied, many of the wavelets exhibit densities that are below that of the time domain.

The threshold and Jacobian density at each iteration with the Haar wavelet is shown in Figure 4.24. There are no large variations in Jacobian density observed as the threshold changes. The Jacobian thresholding method is successful in keeping the error introduced by thresholding the Jacobian matrix low enough not to alter the number of iterations required for the simulation to converge.

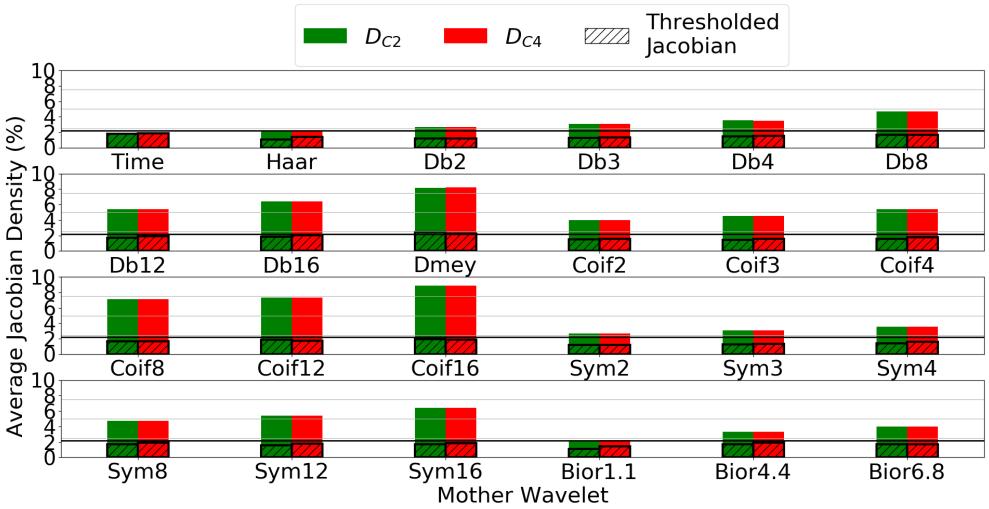

Figure 4.23: Average Jacobian densities for the transmission line circuit.

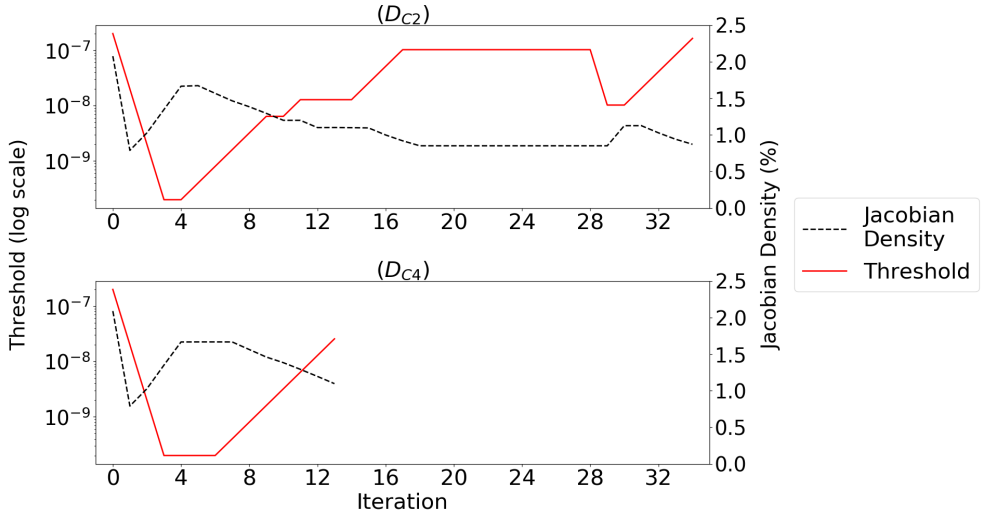

Figure 4.24: Variation of h and Jacobian density at each iteration for the transmission line circuit using the Haar wavelet.

The density of $\hat{\boldsymbol{v}}$ is shown in Figure 4.25. The wavelets with the lowest number of vanishing moments led to the lowest density $\hat{\boldsymbol{v}}$ with all but the biorthogonal wavelets. Jacobian thresholding had no significant effect on the density of $\hat{\boldsymbol{v}}$.

The number of iterations for each of the simulations is shown in Figure 4.26. Thresholding caused no change in the number of iterations required for convergence. The number of iterations was often not significantly changed when using wavelets instead of the time domain. However, for the D_{C4} derivative method and Haar, Bior1.1, and Bior4.4 wavelets, there is a significant difference between the time and wavelet domain.

The simulation run times for the transmission line circuit for the time domain and Haar and Sym2

wavelets are shown in Table 4.7. There is a speedup with all of the adaptively thresholded simulations, compared to SSWA with no thresholding. The static threshold used for this circuit was 10^{-8} which represents a rough average of the threshold values used by the adaptive thresholding method (see Figure 4.24). Based on per-iteration simulation times, all adaptively thresholded simulations but the Db2 wavelet simulations with the D_{C2} derivative ran faster than SSTD. For half of the test cases, the static threshold had lower per-iteration numeric factorization times than the adaptive Jacobian matrix thresholding method.

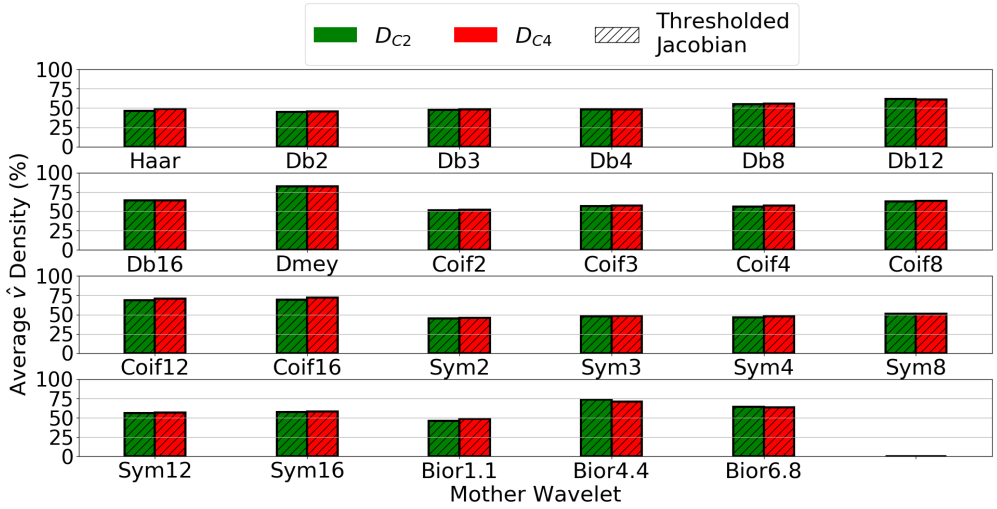

Figure 4.25: Average $\hat{\boldsymbol{v}}$ densities for the transmission line circuit.

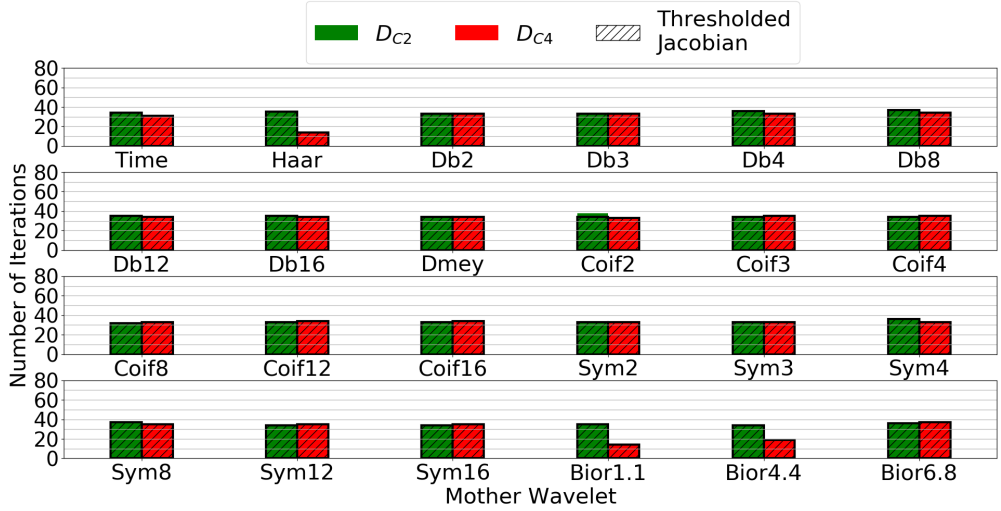

Figure 4.26: Number of iterations for the transmission line circuit.

Table 4.7: Total simulation, symbolic factorization, and numeric factorization timing data, in seconds, for the transmission line circuit with the Haar, Db2, and Sym2 wavelets. The number of iterations is shown in brackets. The static threshold value was 10^{-8}. Green font indicates the cases where the adaptive thresholding method is faster than the non-thresholded method while blue font indicates the cases where the static thresholding method is faster than the adaptive thresholding method.

		D_{C2}			D_{C4}		
		No Thresh	Adaptive	Static	No Thresh	Adaptive	Static
Time	Total	86.41 [2.541]	95.23 [2.801]	115.9 [3.410]	107.6 [3.470]	109.3 [3.528]	109.5 [3.532]
	Symbolic	2.538 [0.075]	2.267 [0.067]	2.578 [0.076]	2.181 [0.070]	2.221 [0.072]	2.223 [0.072]
	Numeric	54.00 [1.588]	63.15 [1.857]	79.69 [2.344]	79.12 [2.552]	77.64 [2.505]	76.06 [2.454]
	Iterations	34	34	34	31	31	31
Haar	Total	98.97 [2.828]	61.04 [1.744]	67.15 [1.919]	52.46 [3.497]	32.42 [2.316]	31.21 [2.080]
	Symbolic	3.069 [0.088]	1.990 [0.057]	2.013 [0.057]	1.478 [0.099]	1.034 [0.074]	0.923 [0.062]
	Numeric	59.55 [1.701]	25.98 [0.742]	28.82 [0.823]	33.51 [2.234]	15.21 [1.087]	13.16 [0.875]
	Iterations	35	35	35	15	14	15
Db2	Total	86.89 [2.633]	85.59 [2.594]	90.78 [2.751]	86.91 [2.634]	84.66 [2.565]	87.27 [2.645]
	Symbolic	4.183 [0.127]	3.776 [0.114]	4.141 [0.126]	4.312 [0.131]	3.811 [0.116]	4.149 [0.126]
	Numeric	43.40 [1.315]	42.67 [1.293]	44.17 [1.338]	42.72 [1.295]	41.72 [1.264]	41.82 [1.267]
	Iterations	33	33	33	33	33	33
Sym2	Total	89.77 [2.720]	87.79 [2.660]	84.15 [2.550]	89.99 [2.727]	83.88 [2.542]	83.82 [2.540]
	Symbolic	4.465 [0.135]	3.807 [0.115]	3.847 [0.117]	4.435 [0.134]	3.806 [0.115]	3.915 [0.119]
	Numeric	43.98 [1.333]	44.61 [1.352]	41.10 [1.246]	45.86 [1.390]	41.69 [1.263]	40.46 [1.226]
	Iterations	33	33	33	33	33	33

4.2.5 Stepped Impedance Filter Circuit

The last case study is the stepped impedance filter circuit shown in Figure 4.27. There are 64 nodal variables in this circuit with 512 time samples which results in 32768 variables, making this the largest problem tested in this study. The transmission lines are modelled in the frequency domain using scattering parameters that are generated using their physical characteristics which are given in Figure 4.27. The filter has a cutoff frequency of 14Ghz. The LMA411 is a model of the Filtronics X-band MMIC low noise amplifier[60]. The non-thresholded (solid line) and thresholded (dashed line) input and output waveforms

and the voltage at the input of the LMA411 (node A) are shown in Figure 4.28.

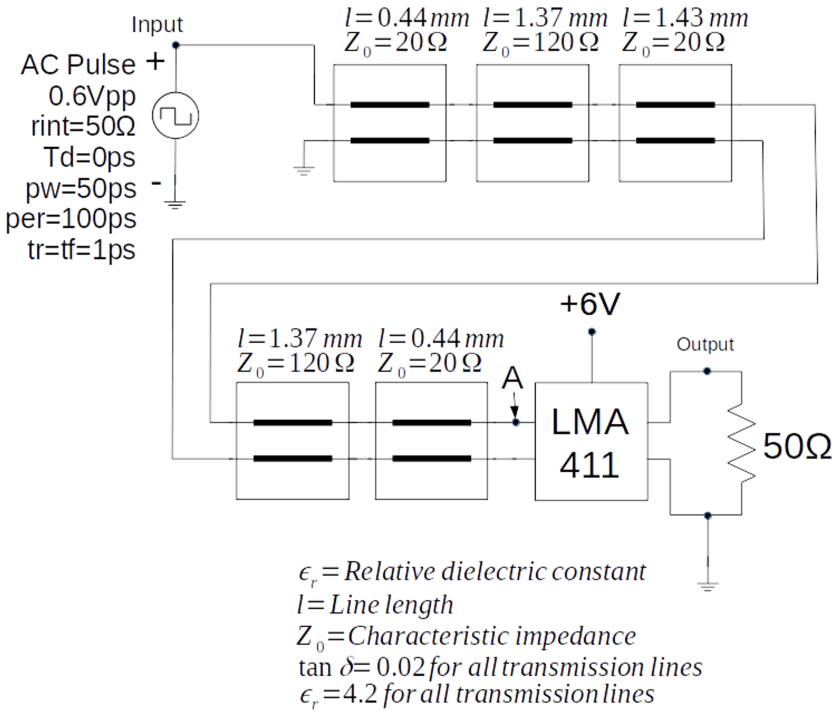

Figure 4.27: Stepped impedance filter circuit schematic.

The Jacobian densities are given in Figure 4.29. The wavelets with the lowest number of vanishing moments provide the lowest density Jacobian matrices while the greatest reduction in density occurs with wavelets that have the highest number of vanishing moments. When Jacobian thresholding is applied, many of the wavelets exhibit densities that are below that of the time domain.

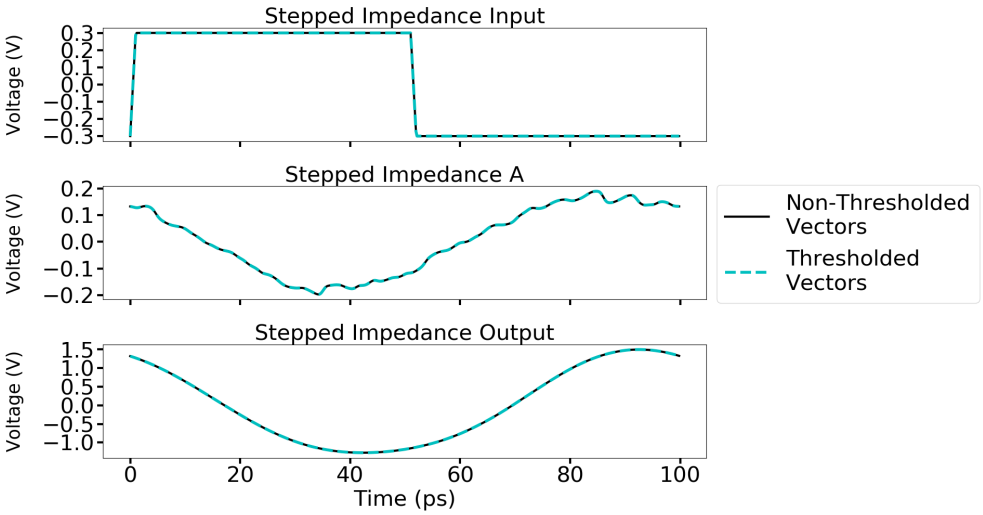

Figure 4.28: Input and output waveforms for the stepped impedance filter circuit. Dashed line represents the signals after they are thresholded.

Figure 4.30 shows the density of $\hat{\boldsymbol{v}}$. Due to the presence of very smooth waveforms, the wavelets with the lowest number of vanishing moments do not tend to provide the lowest density $\hat{\boldsymbol{v}}$ waveforms. The lowest density $\hat{\boldsymbol{v}}$ occurs with the Db4 wavelet.

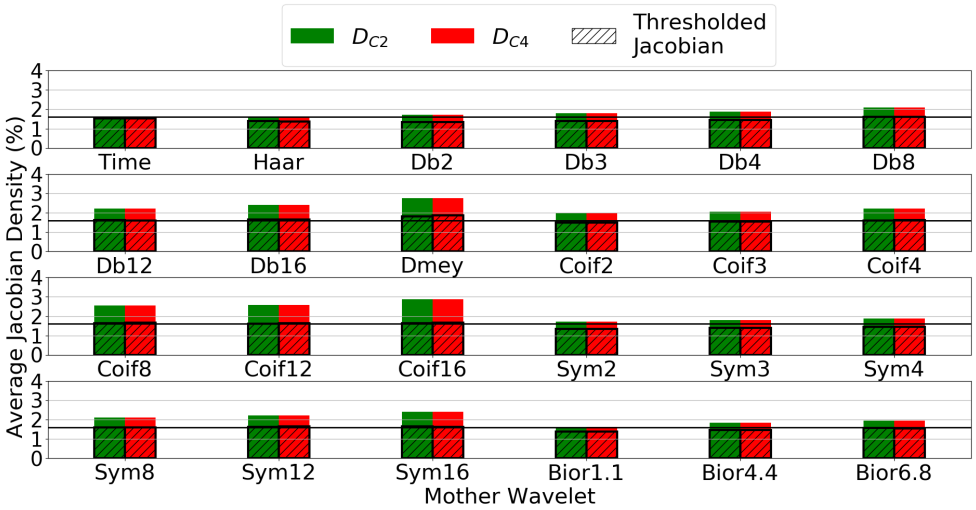

Figure 4.29: Average Jacobian densities for the stepped impedance filter circuit.

The variation of h and the Jacobian density with the Haar wavelet are shown in Figure 4.31. There are no large variations in Jacobian density observed as the threshold changes.

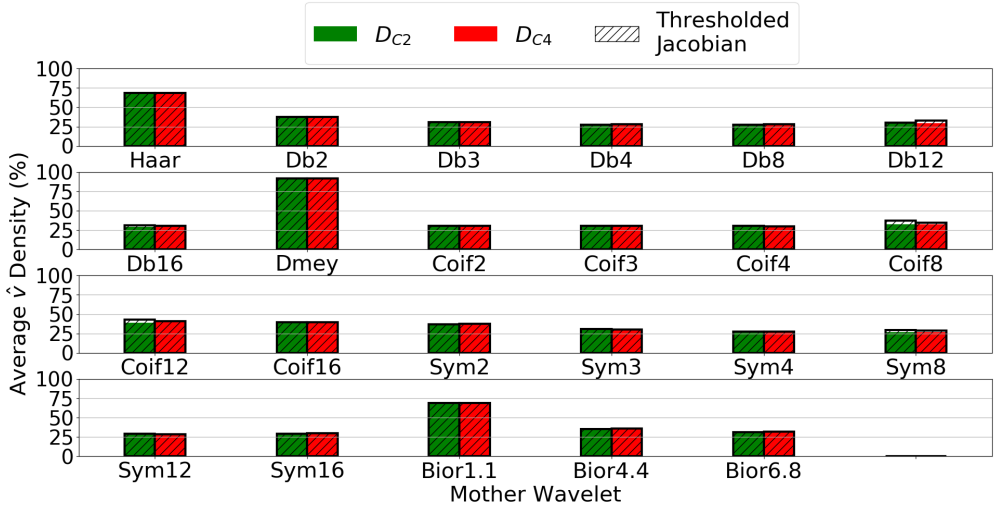

Figure 4.30: Average $\hat{\boldsymbol{v}}$ densities for the stepped impedance filter circuit.

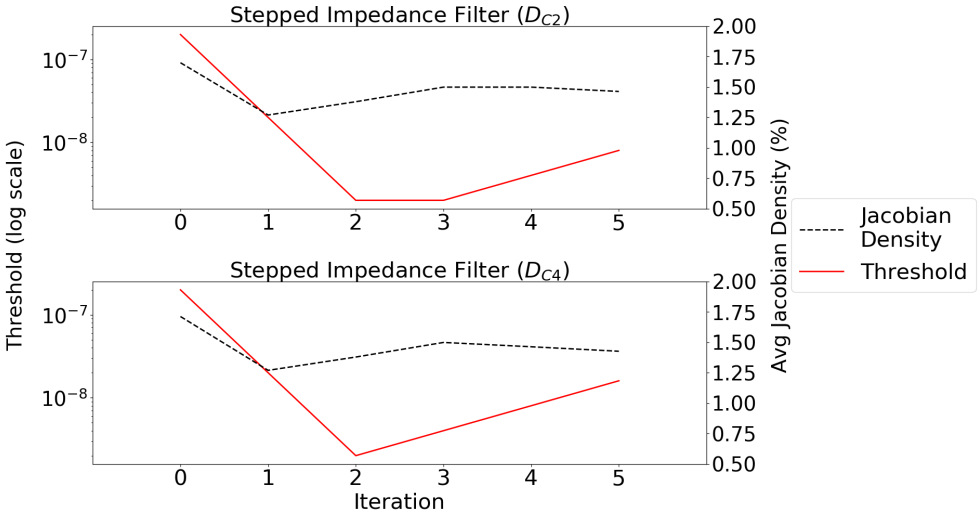

Figure 4.31: Variation of h and Jacobian density at each iteration for the stepped impedance filter circuit using the Haar wavelet.

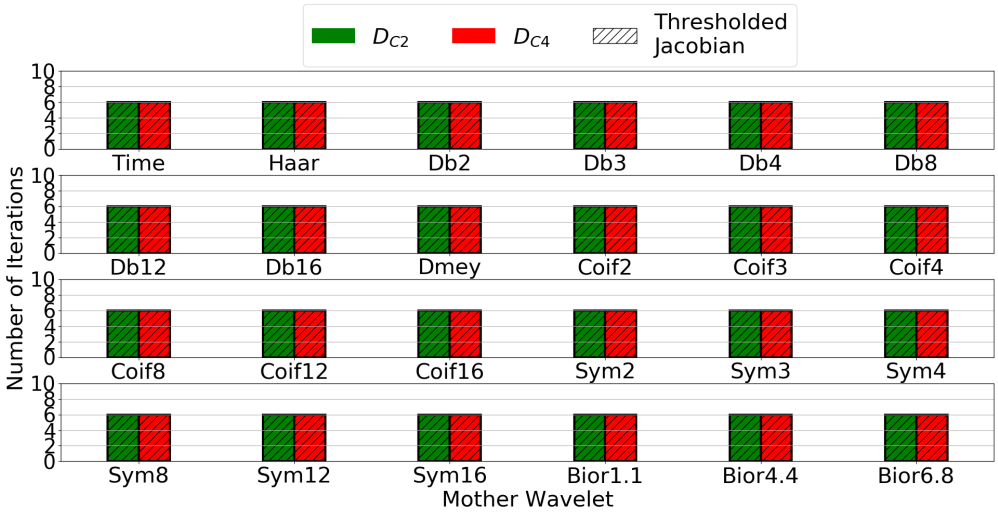

Figure 4.32: Number of iterations for the stepped impedance filter circuit.

The number of iterations for each of the simulations is shown in Figure 4.32. Thresholding caused no change in the number of iterations required for convergence. The number of iterations was not changed when using wavelets instead of the time domain. The simulation run times for the stepped impedance filter circuit for the time domain and Haar and Db4 wavelets are shown in Table 4.8. The static threshold used for this circuit was 10^{-8} which represents a rough average of the threshold values used by the adaptive thresholding method (see Figure 4.31). There is a speedup with the Haar wavelet when adaptive Jacobian thresholding is applied. This speedup is enough for the adaptively thresholded simulations to be faster than SSTD when SSWA with no thresholding was not. For half of the test cases, the static threshold led to a lower per-iteration numerical factorization time than the adaptive thresholding method.

Table 4.8: Total simulation, symbolic factorization, and numeric factorization timing data, in seconds, and calculated speedup for the stepped impedance filter circuit with the Haar and Db4 wavelets. The number of iterations is shown in brackets. The static threshold value was 10^{-8}. Green font indicates the cases where the adaptive thresholding method is faster than the non-thresholded method while blue font indicates the cases where the static thresholding method is faster than the adaptive thresholding method.

		D_{C2}			D_{C4}		
		No Thresh	Adaptive	Static	No Thresh	Adaptive	Static
Time	Total	287.5 [47.91]	272.9 [45.48]	288.8 [48.13]	292.2 [48.71]	284.6 [47.44]	291.4 [48.56]
	Symbolic	5.455 [0.909]	5.323 [0.887]	5.365 [0.894]	5.470 [0.912]	5.074 [0.846]	4.952 [0.825]
	Numeric	246.4 [41.06]	230.4 [38.40]	245.3 [40.89]	249.1 [41.52]	241.6 [40.27]	247.0 [41.16]
	Iterations	6	6	6	6	6	6
Haar	Total	328.1 [54.68]	257.7 [42.95]	245.6 [40.93]	329.0 [54.83]	272.5 [45.41]	251.6 [41.93]
	Symbolic	5.789 [0.965]	5.069 [0.845]	5.253 [0.876]	5.585 [0.930]	5.475 [0.912]	5.233 [0.872]
	Numeric	279.6 [46.60]	207.5 [34.58]	197.8 [32.97]	277.7 [46.28]	222.7 [37.12]	203.94 [33.99]
	Iterations	6	6	6	6	6	6
Db4	Total	267.6 [44.61]	350.9 [58.48]	339.42 [56.57]	264.7 [44.11]	345.0 [57.50]	341.3 [56.88]
	Symbolic	12.71 [2.119]	11.73 [1.956]	11.88 [1.980]	15.588 [2.598]	13.78 [2.296]	12.58 [2.10]
	Numeric	193.4 [32.24]	275.5 [45.92]	258.5 [43.09]	187.8 [31.30]	261.5 [43.59]	263.7 [43.95]
	Iterations	6	6	6	6	6	6

4.3 Discussion

The Newton updates are calculated by performing LU factorization on the thresholded Jacobian matrices using the UMFPack routines in SuiteSparse [49]. The difference in average density of the L and U matrices between non-thresholded SSWA and SSWA with adaptive Jacobian matrix thresholding are shown in Figures 4.33 and 4.34. Since the adaptive thresholding method will have an effect on the sparsity pattern of the Jacobian matrix, the densities of the L and U matrices are compared to determine the effect the adaptive Jacobian matrix thresholding method has on fill-in. The difference in non-zero elements is calculated by subtracting the number of non-zero elements in the L and U matrices of SSWA with adaptive Jacobian matrix thresholding from the number of non-zero elements in the L and U matrices of SSWA without Jacobian matrix thresholding. Thus, a positive difference in number of non-zero elements indicates that there are fewer non-zero elements in the L and U matrices when adaptive Jacobian matrix thresholding is applied. The average density of the L and U matrices is lower when adaptive Jacobian

matrix thresholding is applied in all cases for the inverter chain, Gilbert cell, and ultrasound transducer driver circuits. However, some wavelets with the transmission line circuit exhibit an increase in L matrix density and many wavelets with the stepped impedance filter circuit exhibit an increase in L and/or U densities. Thus it is possible for fill-in to increase, and negatively affect simulation speed and memory utilization, with the adaptive thresholding method if the first iteration column pre-ordering is re-used.

Figure 4.33: Difference between the average density of the L matrices with and without adaptive Jacobian matrix thresholding. A positive number indicates that the L matrices resulting from the non-thresholded Jacobian matrices have a higher number of non-zero elements, on average, than the L matrices resulting from the adaptively thresholded Jacobian matrices.

Figure 4.34: Difference between the average density of the U matrices with and without adaptive Jacobian matrix thresholding. A positive number indicates that the U matrices resulting from the non-thresholded Jacobian matrices have a higher number of non-zero elements, on average, than the U matrices resulting from the adaptively thresholded Jacobian matrices.

With the stepped impedance filter circuit, there is a slowdown when the Db4 wavelet was utilized (see Figure 4.8). The increase in average L and U matrix densities when adaptive Jacobian matrix thresholding is utilized is high which indicates a significant increase in computational resources required to compute the extra elements, on average, in the L and U matrices and leads to the slowdown seen with this test case. For the Db2 and Sym2 wavelets with the transmission line with inverter circuit, there is an increase in L density but there is a decrease in U density. The increase in L density is high enough to remove most of the advantage of thresholding the Jacobian matrix and the speedup with these wavelets is very

small as a result. The increase in L and U matrices appears to be the cause of the slowdown exhibited with the Db4 wavelet with the stepped impedance filter circuit.

Since the Jacobian matrix sparsity pattern can change each iteration as the adaptive threshold changes, it is possible for the sparse matrix pre-ordering found at the first iteration to become less efficient for minimizing fill-ins. To test if a static threshold can mitigate this problem, the simulation speeds of a static threshold were compared with the adaptive thresholding method. The results show for many, but not all, cases the adaptive threshold leads to faster per-iteration numeric factorization times than the static threshold. For the stepped impedance filter, the fill-ins caused a slowdown with the adaptive threshold method. The static threshold exhibited lower factorization times for some of the test cases with the stepped impedance filter circuit but these factorization times were still higher than the non-thresholded simulations. It is possible that, by experimenting with thresholds, the static threshold method could lead to faster simulation and factorization times than the adaptive thresholding method. However, one of the problems that the adaptive thresholding method addresses is the need for a designer to manually determine an ideal threshold for the simulation. Thus, using the adaptive thresholding method removes the need for selecting an ideal threshold but comes at a trade off of a possible slowdown compared to an ideally selected static threshold.

The adaptive threshold succeeded in reducing the average Jacobian density in the cases tested without significantly altering the total number of iterations. The reduction in Jacobian density is higher for wavelets that have a higher number of vanishing moments but the wavelets with the lowest number of vanishing moments operate with the lowest Jacobian densities of the wavelet families both with and without adaptive thresholding. The time domain tends to have the lowest Jacobian densities for circuits without frequency defined components but, with thresholding, it is possible for the wavelets to achieve lower Jacobian densities than the time domain when frequency defined components are present.

The average density of $\hat{\boldsymbol{v}}$ for all iterations was presented to estimate the potential savings that could be achieved by methods that take advantage of low density nodal vectors. This density tends to be lowest with wavelets that have the lowest number of vanishing moments. However, the Haar wavelet, which provided the lowest density Jacobian matrices when Jacobian thresholding was not applied, exhibits a higher $\hat{\boldsymbol{v}}$ density over other wavelets that have the a similar number of vanishing moments (Db2, Coif2, and Sym2). This is due to the Haar wavelet not being able to describe smoothly changing waveforms as sparsely as the other wavelet families. Thus, in terms of vector sparsity, Db2, Coif2, and Sym2 wavelets are better suited than the Haar wavelet if a method that takes advantage of the sparsity of nodal variable vectors is being used and the Sym2 wavelet provides the sparsest $\hat{\boldsymbol{v}}$ vectors of these wavelet families. For the stepped impedance filter circuit the sparsest vector representations were not provided by the wavelets

with the lowest number of vanishing moments. Thus the Haar, Db2, Coif2, and Sym2 wavelets do not always provide the sparsest vectors and simply selecting a wavelet with the lowest number of vanishing moments will not guarantee the sparsest $\hat{\boldsymbol{v}}$ vectors will be achieved. The density of the $\hat{\boldsymbol{v}}$ vectors is much lower for all wavelet domain representations than the time domain.

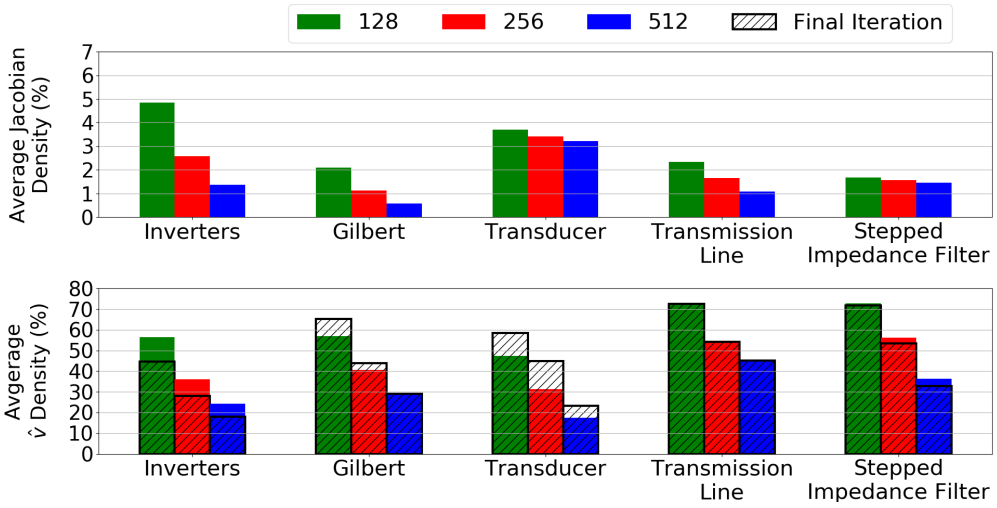

Figure 4.35: Effect of number of time samples on average Jacobian matrix density and average $\hat{\boldsymbol{v}}$ density.

The effect of changing the number of time samples on the average densities of the Jacobian matrix for the Haar wavelet and $\hat{\boldsymbol{v}}$ using the Sym2 wavelet with the D_{C2} derivatives are shown in Figure 4.35 (results for D_{C4} derivatives are similar). The shaded boxes indicate the $\hat{\boldsymbol{v}}$ densities for the final results. The average density of the Jacobian matrix is approximately halved as the number of samples is doubled for the circuits without frequency defined elements. However, when frequency defined elements are introduced, the density reduction rate is lower. The density of the nodal variable vectors does not follow the same trend and is approximately halved for the inverter chain and transducer driver while the reduction rate is lower for the Gilbert cell, transmission line, and stepped impedance filter circuits.

4.4 Chapter Summary

This chapter presented a study on the effect of wavelet selection on the sparsity of the Jacobian matrices and the Newton update vectors of periodic steady-state analysis with wavelets. Also presented was a method that increased the sparsity of the Jacobian matrix each iteration via an adaptive threshold. Based on this data, Jacobian matrix densities tends to be higher with wavelets than the time domain, even with adaptive Jacobian matrix thresholding, in the majority of cases.

The reduction in densities tends to be highest with wavelets that have a higher number of vanishing moments but this reduction is not high enough to result in Jacobian matrix densities that are as low as wavelets with a lower number of vanishing moments. When not utilizing adaptive thresholding, the Haar

wavelet provided the sparsest Jacobian matrices of the wavelets tested. Even though it is possible for other wavelets to achieve a greater sparsity than the Haar wavelet when Jacobian thresholding is applied, it is safe to say that, if unsure of which wavelet to select, the Haar wavelet should be selected if Jacobian matrix sparsity is the main concern.

Adaptive Jacobian matrix thresholding successfully increases the speed of SSWA for nearly every test case. In some cases, the speedup was high enough to cause the simulations to be faster than SSTD when the non-thresholded SSWA simulations did not. It is also possible for the number of iterations to be altered when applying adaptive Jacobian matrix thresholding. This change in number of iterations could potentially be addressed by improvement of error control with adaptive Jacobian matrix thresholding. However, in its present form, the adaptive Jacobian matrix thresholding method is useful for increasing the efficiency of SSWA. For two of the test circuits, there was a small increase in speed when applying adaptive Jacobian matrix thresholding to SSTD. However, given the change in Jacobian matrix densities was not significant with the SSTD simulations, it does not seem that there is much, if any, advantage to applying adaptive Jacobian matrix thresholding to SSTD.

The average density of $\hat{\boldsymbol{v}}$ was significantly lower than the average density of $\bar{\boldsymbol{v}}$ in the majority of cases, which implies that there may be greater potential to reduce the computational complexity of SSWA by taking advantage of the wavelet domain vector sparsity. Even wavelets that exhibited very low reductions in Jacobian sparsity with adaptive thresholding tended to have much lower average $\hat{\boldsymbol{v}}$ densities than SSTD. Therefore, a method that takes advantage of the sparse representations of $\hat{\boldsymbol{v}}$ has the potential to achieve more consistent gains in efficiency at each iteration. For this reason, two algorithms were developed that reduce the number of columns in the Jacobian matrix at each iteration of Newton method. These algorithms are presented and tested in the next chapter.

Chapter 5

Steady-State Analysis of Electronic Circuits Using Wavelets with Adaptive Jacobian Columns

In this chapter, two methods called Reduced Column Steady-State Wavelet Analysis (RCSSWA) and Hybrid Reduced Column Steady-State Wavelet Analysis (HRCSSWA) are presented along with the results of simulations run with several test circuits.

RCSSWA takes advantage of the sparse wavelet representations of the nodal variable waveforms to attempt to reduce the number of columns in the Jacobian matrix during each iteration of Newton method. The reduced Jacobian is a tall rectangular matrix that has a lower number of non-zero elements than the original Jacobian matrix on iterations where the column reduction is performed. The column selection criterion is based on the error introduced into the computations and, by keeping the introduced error at a minimum, it is possible to achieve a significant reduction in the number of columns and average number of non-zero elements in the Jacobian matrix compared to SSWA. HRCSSWA combines RCSSWA and SSTD into a hybrid wavelet-time method that utilizes a reduced column wavelet transform for the formation of a reduced column wavelet domain Jacobian matrix. The reduced Jacobian matrices with HRCSSWA have a lower number of non-zero elements, and require less computational overhead to form, than the SSWA and RCSSWA Jacobian matrices.

5.1 Reduction of Jacobian Matrix Columns

The RCSSWA method takes advantage of the ability of wavelets to sparsely represent a given waveform to find a sparse estimate of the solution of Equation (2.58). This is achieved by creating a list, S_{Δ}^{j}, of the positions of the non-zero elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$. This list, which is also known as the support of $\Delta\hat{\boldsymbol{v}}^{j+1}$, is

used to create a Jacobian matrix with a reduced number of columns which is used to calculate a sparse estimate of $\Delta\hat{\boldsymbol{v}}^{j+1}$ at each iteration. Each element in $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$ is related to $\Delta\hat{\boldsymbol{v}}^{j+1}$ by expressing Equation (2.58) as the sum:

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)_k = -\sum_{l=0}^{NM} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}\Delta\hat{\boldsymbol{v}}_l^{j+1}. \tag{5.1}$$

Since S_Δ^j is a list of the indices of the non-zero elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$, the above equation can be separated into two sums:

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)_k = -\sum_{l\in S_\Delta^j} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}\Delta\hat{\boldsymbol{v}}_l^{j+1} - \sum_{l\notin S_\Delta^j} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}\Delta\hat{\boldsymbol{v}}_l^{j+1}. \tag{5.2}$$

Since all elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$ that are not in set S_Δ^j are expected to be zero, the sum over all $l \notin S_\Delta^j$ in Equation (5.2) can be considered to be equal to zero for all k. Thus, a sparse estimate of $\Delta\hat{\boldsymbol{v}}^{j+1}$ can be found by using only the columns in $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ that correspond to elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$ that are in set S_Δ^j.

The sparse estimate of $\Delta\hat{\boldsymbol{v}}^{j+1}$ is given by:

$$\Delta\hat{\boldsymbol{v}}_\Delta^{j+1} = -\hat{\boldsymbol{J}}_\Delta^+\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j), \tag{5.3}$$

where $\hat{\boldsymbol{J}}_\Delta$ is an $NM \times m$ matrix formed from the columns in $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ that correspond to the non-zero elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$, $\hat{\boldsymbol{J}}_\Delta^+$ is the Moore-Penrose inverse of $\hat{\boldsymbol{J}}_\Delta$ given by $\hat{\boldsymbol{J}}_\Delta^+ = (\hat{\boldsymbol{J}}_\Delta^T\hat{\boldsymbol{J}}_\Delta)^{-1}\hat{\boldsymbol{J}}_\Delta^T$, $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$ is a vector of all non-zero elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$, and m represents the number of non-zero elements in $\Delta\hat{\boldsymbol{v}}^{j+1}$.

Selection of elements to include in set S_Δ^j is performed by considering both $\hat{\boldsymbol{v}}^{j+1}$ and $\hat{\boldsymbol{v}}^j$. Substituting $\Delta\hat{\boldsymbol{v}}^{j+1} = \hat{\boldsymbol{v}}^{j+1} - \hat{\boldsymbol{v}}^j$ into Equation(5.1) results in:

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)_k = -\sum_{l=0}^{NM} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}(\hat{\boldsymbol{v}}_l^{j+1} - \hat{\boldsymbol{v}}_l^j). \tag{5.4}$$

Which can be separated into two sums:

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)_k = -\sum_{l=0}^{NM} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}\hat{\boldsymbol{v}}_l^{j+1} + \sum_{l=0}^{NM} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}\hat{\boldsymbol{v}}_l^j. \tag{5.5}$$

Let S_v^{j+1} be a list of the indices of all non-zero elements in $\hat{\boldsymbol{v}}^{j+1}$ and S_v^j be a list of the indices of all non-zero elements in $\hat{\boldsymbol{v}}^j$. Since all elements in $\hat{\boldsymbol{v}}^{j+1}$ that are not indexed by S_v^{j+1} are equal to zero and all elements in $\hat{\boldsymbol{v}}^j$ that are not in S_v^j are equal to zero, Equation (5.5) becomes:

$$\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)_k = -\sum_{l\in S_v^{j+1}} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}\hat{\boldsymbol{v}}_l^{j+1} + \sum_{l\in S_v^j} \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)_{k,l}\hat{\boldsymbol{v}}_l^j. \tag{5.6}$$

Therefore, both S_v^{j+1} and S_v^j are combined to estimate S_Δ^j:

$$S_\Delta^j = S_v^{j+1} \cup S_v^j. \tag{5.7}$$

Since part of S_{Δ}^{j} comes from the known vector $\hat{\boldsymbol{v}}^{j}$, the support estimate will focus on the unknown vector $\hat{\boldsymbol{v}}^{j+1}$. Substituting $\Delta\hat{\boldsymbol{v}}^{j+1} = \hat{\boldsymbol{v}}^{j+1} - \hat{\boldsymbol{v}}^{j}$ into Equation (2.58) and rearranging it to calculate $\hat{\boldsymbol{v}}^{j+1}$ directly yields:

$$\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j}) \cdot \hat{\boldsymbol{v}}^{j+1} = -\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^{j}) + \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j}) \cdot \hat{\boldsymbol{v}}^{j}. \tag{5.8}$$

S_{v}^{j+1} can then be estimated using an approach that is similar to those used by the orthogonal matching pursuit algorithms used with compressed sensing to find sparse estimates of the solution to the system of equations of the form[61]:

$$A_{OMP}x_{OMP} = f_{OMP}, \tag{5.9}$$

where A_{OMP} is a dictionary matrix that transforms the elements given by sparse vector x_{OMP} into the measurement vector f_{OMP}.

Orthogonal matching pursuit algorithms have two fundamental steps: element selection and coefficient update. During element selection, coefficients are selected to be included in the support of x_{OMP}. The elements are selected to minimize the residual given by:

$$r_{OMP} = f_{OMP} - A_{OMP}x_{OMP}. \tag{5.10}$$

To perform element selection, r_{OMP} is projected onto the columns of A_{OMP}:

$$g_{OMP} = A_{OMP}^{T} \cdot r_{OMP}. \tag{5.11}$$

Next, the elements from g_{OMP} with the highest amplitudes are selected and added to S_{OMP} (the support of x_{OMP}).

During the coefficient update step, a matrix is formed from the columns of A_{OMP} that are in the selected support of x_{OMP}. The new matrix is then used to calculate a new x_{OMP}. The element selection and coefficient update steps are repeated until a selected stopping criterion is met. A basic outline of an orthogonal matching pursuit algorithm is shown in Algorithm 5.1.

In this thesis, S_{v}^{j+1} is estimated by substituting $r_{OMP} = -\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^{j}) + \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j}) \cdot \hat{\boldsymbol{v}}^{j}$ and $A_{OMP} = \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j})$ into Equation (5.11) to calculate:

$$\hat{\boldsymbol{g}}_{v} = \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j})^{T} \cdot (-\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^{j}) + \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j}) \cdot \hat{\boldsymbol{v}}^{j}), \tag{5.12}$$

then thresholding the elements in $\hat{\boldsymbol{g}}_{v}$ with the lowest absolute values. Using this column selection method for finding S_{v}^{j+1} can lead to multiple iterations for finding $\hat{\boldsymbol{v}}^{j+1}$ which can potentially be more computationally expensive than simply calculating $\hat{\boldsymbol{v}}^{j+1}$ directly using:

$$\hat{\boldsymbol{v}}^{j+1} = \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j})^{-1} \cdot (-\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^{j}) + \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j}) \cdot \hat{\boldsymbol{v}}^{j}). \tag{5.13}$$

For this reason, all elements in the thresholded $\hat{\boldsymbol{g}}_{v}$ vector are used to form the estimate of S_{v}^{j+1}. This search method can cause errors in the support estimate which can lead to error between $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ and

Algorithm 5.1 Basic orthogonal matching pursuit algorithm. Adapted from [61].

$j \leftarrow 0$

$x_{OMP} \leftarrow$ All entries set to zero.

$S_{OMP} \leftarrow$ All entries set to zero.

$r_{OMP} \leftarrow f_{OMP}$

while $\|r_{OMP}\|_2 <$ selected convergence level **do**

$g_{OMP} = A_{OMP}^T \cdot r_{OMP}$

$S_{OMP} \leftarrow$ Indices of elements in g_{OMP} with amplitudesthat are greater than a selected threshold to S_{OMP}.

$A_{OMP,S} \leftarrow$ Copy of columns from A_{OMP} indexed by S_{OMP}.

$x_{OMP} \leftarrow A_{OMP,S}^+ \cdot f_{OMP}$

$r_{OMP} \leftarrow f_{OMP} - A_{OMP} \cdot x_{OMP}$

end while

$\Delta\hat{\boldsymbol{v}}^{j+1}$. When the error is low, it can be corrected for in a future Newton iteration. When the error is high, it must be corrected when the Newton update vector is calculated. Methods for estimating and correcting the error will be discussed in Section 5.2.

Vector $\hat{\boldsymbol{v}}^j$ contains many very low amplitude coefficients that can be removed without introducing a significant amount of error and low amplitude elements in vector $\hat{\boldsymbol{g}}_v$ also need to be removed to ensure S_Δ^j is as sparse as possible.

The thresholding method used in this paper is based on the highest amplitude coefficient in each nodal waveform. The threshold is automatically selected using a relative tolerance and a noise tolerance. A single threshold value that works well for one nodal waveform will not necessarily work well for another. The relative tolerance addresses this issue by basing the threshold on the highest amplitude wavelet coefficient while the noise tolerance is used to remove numerical noise. For each node, the threshold is selected using the following formula:

$$h = r_{tol}\|\hat{\boldsymbol{z}}_k\|_\infty + n_{tol}, \tag{5.14}$$

where n_{tol} is a selected noise tolerance, r_{tol} is a selected relative tolerance and $\hat{\boldsymbol{z}}_k$ are the wavelet domain coefficients of the waveform for node k of the vector being thresholded. All scaling and wavelet coefficients for node k that have an amplitude less than h are set to zero.

5.2 Reduced Column Steady-State Wavelet Analysis

A method that utilizes the LU decomposition of $\hat{\boldsymbol{J}}_\Delta$ has been selected to provide an estimate of the least squares solution of Equation (5.3). This method finds an estimate of $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$ as well as an error factor

that represents the difference between the estimate of $\Delta\hat{\boldsymbol{v}}^{j+1}$ and $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$. If the error factor is below a reasonable threshold, the estimate of $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ is considered an accurate estimate of $\Delta\hat{\boldsymbol{v}}^{j+1}$. If the error factor is above the threshold, $\Delta\hat{\boldsymbol{v}}^{j+1}$ is calculated using Equation (2.58) instead.

LU decomposition can be applied to find the least squares solution for over determined systems of equations of the form $Ax = b$ by using a pseudo-inverse of A [62]. The pseudo-inverse can be formed using factored matrices with different factoring methods (including LU decomposition). Björck and Duff [63, 64] introduced a partitioning scheme to split the problem into two parts. One part that represents an estimate to the least squares solution and a second part that represents a residual which is the difference between the estimate and least squares solutions.

The LU decomposition of $\hat{\boldsymbol{J}}_{\Delta}$ is given by:

$$\boldsymbol{P}_r\hat{\boldsymbol{J}}_{\Delta}\boldsymbol{P}_c = \boldsymbol{L}\boldsymbol{U}, \tag{5.15}$$

where $\boldsymbol{P}_c$ is an $m \times m$ column permutation matrix, $\boldsymbol{P}_r$ is an $NM \times NM$ row permutation matrix, $\boldsymbol{L}$ is an $NM \times m$ lower trapezoidal matrix, and $\boldsymbol{U}$ is an $m \times m$ upper triangular matrix. From [63, 64], the least squares solution to Equation (5.3) can be posed as:

$$\min_{\hat{\boldsymbol{y}}} \| - \boldsymbol{P}_r\hat{\boldsymbol{f}}\left(\hat{\boldsymbol{v}}^{j}\right) - \boldsymbol{L}\hat{\boldsymbol{y}}\|_2 \qquad s.t.\ \hat{\boldsymbol{y}} = \boldsymbol{U}\boldsymbol{P}_c^T\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}. \tag{5.16}$$

Following the methodology in [63, 64], $\boldsymbol{L}$, $-\hat{\boldsymbol{f}}\left(\hat{\boldsymbol{v}}^{j}\right)$, and $\hat{\boldsymbol{y}}$ are partitioned into two parts:

$$\boldsymbol{L} = \begin{bmatrix}\boldsymbol{L}_1 \\ \hline \boldsymbol{L}_2\end{bmatrix}, \quad -\boldsymbol{P}_r\hat{\boldsymbol{f}}\left(\hat{\boldsymbol{v}}^{j}\right) = \begin{bmatrix}\hat{\boldsymbol{f}}_1 \\ \hline \hat{\boldsymbol{f}}_2\end{bmatrix}, \quad \hat{\boldsymbol{y}} = \begin{bmatrix}\hat{\boldsymbol{y}}_1 \\ \hline \hat{\boldsymbol{y}}_2\end{bmatrix}, \tag{5.17}$$

where $\boldsymbol{L}_1$ is an $m \times m$ lower triangular matrix, $\boldsymbol{L}_2$ is an $(NM - m) \times m$ matrix, $\hat{\boldsymbol{f}}_1$ is a vector with m entries, $\hat{\boldsymbol{f}}_2$ is a vector of $NM - m$ entries, $\hat{\boldsymbol{y}}_1$ is a vector with m entries, and $\hat{\boldsymbol{y}}_2$ is a vector of $NM - m$ entries. This partitioning leads to:

$$\boldsymbol{L}_1\hat{\boldsymbol{y}}_1 = \hat{\boldsymbol{f}}_1, \quad \boldsymbol{U}\boldsymbol{P}_c^T\Delta\hat{\boldsymbol{v}}_g = \hat{\boldsymbol{y}}_1, \tag{5.18}$$

where $\Delta\hat{\boldsymbol{v}}_g$ is an m length vector that represents the estimate of $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ and

$$\hat{\boldsymbol{d}}_{\Delta} = \hat{\boldsymbol{f}}_2 - \boldsymbol{L}_2\hat{\boldsymbol{y}}_1, \tag{5.19}$$

where $\hat{\boldsymbol{d}}_{\Delta}$ is a vector with $NM - m$ entries that represents the residual that is associated with the unused part of the partitioned LU factorization in Equation (5.17) and the residual norm corresponding to $\Delta\hat{\boldsymbol{v}}_g$ is:

$$\|\hat{\boldsymbol{d}}_{\Delta}\|_2 = \| - \hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^{j}) - \hat{\boldsymbol{J}}_{\Delta}\Delta\hat{\boldsymbol{v}}_g\|_2. \tag{5.20}$$

For more detail on the derivation of this formula, see Appendix B.

If the residual norm, which represents the error between $\Delta\hat{\boldsymbol{v}}_g$ and $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$, is below an acceptable level, β, then $\Delta\hat{\boldsymbol{v}}_g$ can be considered a solution to Equation (5.3). Otherwise, further refinement of $\Delta\hat{\boldsymbol{v}}_g$ is necessary [63, 64]. Equation (5.20) will be affected by error between $\Delta\hat{\boldsymbol{v}}_g$ and $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$ caused by the LU factorization method as well as errors in the estimate of S_v^{j+1}.[I] Thus, both errors arising from the estimate of S_v^{j+1} as well as errors cause by using $\Delta\hat{\boldsymbol{v}}_g$ can be corrected by monitoring $\|\hat{\boldsymbol{d}}_\Delta\|_2$.

It is possible for $\|\hat{\boldsymbol{d}}_\Delta\|_2$ to be on the same order of magnitude as $\|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\|_2$ when the simulation is close to convergence. When this occurs, the error that results from using $\Delta\hat{\boldsymbol{v}}_g$ can become high enough to prevent the simulation from converging. As a result, it is necessary to ensure that $\|\hat{\boldsymbol{d}}_\Delta\|_2 < \|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\|_2$ so that the error does not prevent convergence. Therefore there are two requirements for $\|\hat{\boldsymbol{d}}_\Delta\|_2$:

$$\|\hat{\boldsymbol{d}}_\Delta\|_2 < \alpha\|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\|_2 \quad \text{and} \quad \|\hat{\boldsymbol{d}}_\Delta\|_2 < \beta, \tag{5.21}$$

where α is a constant that is used to control the acceptable amplitude for the residual norm. If the conditions in Equation (5.21) are satisfied then $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1} = \Delta\hat{\boldsymbol{v}}_g$ is considered an accurate estimate of $\Delta\hat{\boldsymbol{v}}^{j+1}$. Otherwise $\Delta\hat{\boldsymbol{v}}_g$ is further refined.

It is not possible to estimate in advance the densities of the $\boldsymbol{L}_1$, $\boldsymbol{L}_2$, and $\boldsymbol{U}$ matrices. However, the upper limit of their densities can be estimated. Since $\boldsymbol{L}_1$ and $\boldsymbol{U}$ are $m \times m$ lower triangular and upper triangular matrices, their maximum possible densities will be $\left(\frac{1}{m^2}\sum_{i=1}^{m} i\right) * 100\%$ which rapidly approaches 50% as m is increased. The maximum density of the $\boldsymbol{L}_2$ matrix is 100%. If $m << MN$, $\boldsymbol{L}_2$ will contain most of the elements in $\boldsymbol{L}$ and, therefore, can potentially require much higher storage space than $\boldsymbol{L}_1$ and $\boldsymbol{U}$.

Refinement of $\Delta\hat{\boldsymbol{v}}_g$ can be achieved by utilizing the unused portion of the partitioned LU factorization in Equation (5.17). However, this requires utilizing $\hat{\boldsymbol{d}}_\Delta$ and the $\boldsymbol{L}_2$ matrix [63, 64]. There is no guarantee that $\boldsymbol{L}_2$ will not have a greater number of non-zero elements than $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ and it is possible that calculating the update could require more computational time than solving Equation (2.58) directly.

Since $\|\hat{\boldsymbol{d}}_\Delta\|_2$ can be calculated without using $\boldsymbol{L}_2$ and the column reduction method does not perform any refinement using $\hat{\boldsymbol{d}}_\Delta$, the $\boldsymbol{L}_2$ matrix is not required. With careful design of the LU decomposition algorithm, it may be possible to avoid forming the $\boldsymbol{L}_2$ matrix and, therefore, the density of the $\boldsymbol{L}_2$ matrix would not be a concern.

From Equation (5.19), it can be seen that the error in $\Delta\hat{\boldsymbol{v}}_g$ is related to the unused part of the partitioned LU factorization in Equation (5.17). Increasing the number of entries in S_Δ^j will, in turn, increase the number of elements in $\Delta\hat{\boldsymbol{v}}_g$, $\hat{\boldsymbol{y}}_1$, and $\hat{\boldsymbol{f}}_1$ while reducing the number of elements in $\hat{\boldsymbol{y}}_2$ and

[I] This can be proven by substituting $f_{OMP} = -\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$, $A_{OMP} = \hat{\boldsymbol{J}}_\Delta$, and $x_{OMP} = \Delta\hat{\boldsymbol{v}}^{j+1}$ into Equation (5.10). The squared norm of the resulting equation, $\| - \hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) - \hat{\boldsymbol{J}}_\Delta\Delta\hat{\boldsymbol{v}}^{j+1}\|_2$, is the same as Equation (5.20) when $\Delta\hat{\boldsymbol{v}}^{j+1} = \Delta\hat{\boldsymbol{v}}_g$.

$\hat{\boldsymbol{f}}_2$. This, in turn, reduces the effect of the unused part of the partitioned LU factorization in Equation (5.17), increases the accuracy of $\Delta\hat{\boldsymbol{v}}_g$, and reduces $\|\hat{\boldsymbol{d}}_\Delta\|_2$.

Algorithm 5.2 RCSSWA Algorithm

$j \leftarrow 0$

$\hat{\boldsymbol{v}}^0 \leftarrow$ Initial guess.

$\Delta\hat{\boldsymbol{v}}^1 \leftarrow -\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^0)^{-1}\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^0)$

$\hat{\boldsymbol{v}}^1 \leftarrow \hat{\boldsymbol{v}}^0 + \Delta\hat{\boldsymbol{v}}^1$

repeat

$j \leftarrow j + 1$

$\Delta\hat{\boldsymbol{v}}^{j+1} \leftarrow \hat{\boldsymbol{0}}$

$\hat{\boldsymbol{g}}_v \leftarrow \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)^T \cdot (-\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) + \hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j) \cdot \hat{\boldsymbol{v}}^j)$

$S^j_\Delta \leftarrow \text{Support}(\hat{\boldsymbol{g}}_v) \cup \text{Support}(\hat{\boldsymbol{v}}^j)$

$\hat{\boldsymbol{J}}_\Delta \leftarrow$ copy of all columns from $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ in set S^j_Δ

// Calculate $\Delta\hat{\boldsymbol{v}}_g$ with LU decomposition.

$\Delta\hat{\boldsymbol{v}}_g \leftarrow \text{LUSolve}\left(\hat{\boldsymbol{J}}_\Delta, -\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\right)$

$\|\hat{\boldsymbol{d}}_\Delta\|_2 \leftarrow \| - \hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) - \hat{\boldsymbol{J}}_\Delta \cdot \Delta\hat{\boldsymbol{v}}_g\|_2$

if $\left(\|\hat{\boldsymbol{d}}_\Delta\|_2 > \beta\right)$ **or** $\left(\|\hat{\boldsymbol{d}}_\Delta\|_2 > \alpha \cdot \|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\|_2\right)$ **then**

// The conditions in Equation (5.21) are not met,

// calculate $\Delta\hat{\boldsymbol{v}}^{j+1}$ using $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$

// intsead.

$\Delta\hat{\boldsymbol{v}}^{j+1} \leftarrow -\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)^{-1}\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$

else

$\Delta\hat{\boldsymbol{v}}^{j+1}_\Delta \leftarrow \Delta\hat{\boldsymbol{v}}_g$

end if

// Update Guess

$\hat{\boldsymbol{v}}^{j+1} \leftarrow \hat{\boldsymbol{v}}^j + \Delta\hat{\boldsymbol{v}}^{j+1}$

$\hat{\boldsymbol{n}} \leftarrow r_{tol} \min\{|\hat{\boldsymbol{v}}^j|, |\hat{\boldsymbol{v}}^{j+1}|\} + a_{tol}\hat{\boldsymbol{u}}$

until $\Delta\hat{\boldsymbol{v}}^{j+1}_k < \hat{\boldsymbol{n}}_k\,, \forall k \in \{1, 2, \ldots, NM\}$ **and** $\|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^{j+1})\|_\infty < a_{tol}$

Refinement may require multiple steps to lower $\|\hat{\boldsymbol{d}}_\Delta\|_2$ to an acceptable level. Multiple refinement steps can potentially cause a high increase in the overhead of using RCSSWA. Therefore, to keep the number of refinement steps to a minimum, $\Delta\hat{\boldsymbol{v}}^{j+1}$ is calculated using Equation (2.58) when the conditions in Equation (5.21) are not met.

Combining the Jacobian matrix column reduction method in Section 5.1 and the LU decomposition for tall rectangular matrices with Newton method results in the RCSSWA algorithm shown in Algorithm 5.2. In Algorithm 5.2, $\hat{\boldsymbol{u}}$ is a vector with the same number of elements as $\hat{\boldsymbol{v}}^j$ with all elements set to 1, Support($\hat{\boldsymbol{g}}_v$) is the list of all non-zero elements left in $\hat{\boldsymbol{g}}_v^j$ after it has been thresholded using h, Support($\hat{\boldsymbol{v}}^j$) is the list of all non-zero elements left in $\hat{\boldsymbol{v}}^j$ after it has been thresholded using h, and LUSolve $\left(\hat{\boldsymbol{J}}_\Delta, \hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\right)$ is a function that performs the LU factorization described in Section 5.2 on $\hat{\boldsymbol{J}}_\Delta$ and calculates $\Delta\hat{\boldsymbol{v}}_g$ using L_1, U, and $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$.

5.2.1 Simulation Studies

The results of simulations run using the Cardoon circuit simulator for the circuits studied in Section 4.2 are presented in this section. Simulations using the RCSSWA method and an un-modified Steady-State Wavelets Analysis (SSWA) method were performed using wavelets with a low number of vanishing moments from different families as well as the wavelets that provided the sparsest $\hat{\boldsymbol{v}}$ vectors while the derivatives were approximated using the second order central difference formula. The wavelets used in this study are the Haar, Db2, Coif2, Sym2, Sym8, Bior4.4, and Bior6.8 wavelets. The sparsest $\hat{\boldsymbol{v}}$ vectors were determined using the data from the study in Section 4.2. The sparsest wavelet for the inverter chain circuit was the Sym3 wavelet. The Db2 and Sym2 wavelets were tied for the Gilbert cell, ultrasound transducer driver, and transmission line circuits. The Db4 wavelet provided the sparsest $\hat{\boldsymbol{v}}$ vectors for the stepped impedance filter circuit. The simulations were performed with 512 coefficients, $a_{tol} = 10^{-7}$, $n_{tol} = 10^{-15}$, and $r_{tol} = 10^{-4}$. LU factorization of the Jacobian matrices for both SSWA and RCSSWA were performed using the UMFPack routines in SuiteSparse with row scaling disabled [I]. On the first iteration, the pre-ordering that is determined before symbolic decomposition is stored and, on iterations where the reduced column Jacobian matrix is not used, the pre-ordering from the first iteration is re-used for factorization of the full Jacobian matrix. Since columns are selected from the Jacobian matrix to form the reduced column Jacobian matrix, the sparse matrix pre-ordering cannot be re-used with the reduced column Jacobian matrix.

The RCSSWA method is compared with SSWA using the following metrics: simulation run times, calculation times, number of iterations, average number of non-zero elements in the Jacobian matrices, and percentage of non-zero entries in S_Δ^j each iteration. The average densities of the L_1, U, and L_2 matrices and the simulation times for Steady-State Time Domain (SSTD) are also presented. Simulation run times are measured as the total time the simulation took to converge on a solution while calculation times are the total time spent calculating the Newton updates. For SSWA, the calculation time is given

[I] UMFPACK_SCALE=0

by:

$$T_c = \sum_j T^j_{\Delta\hat{v}}, \tag{5.22}$$

where $T^j_{\Delta\hat{v}}$ is the time taken to find $\Delta\hat{\boldsymbol{v}}^{j+1}$ using Equation (2.58). For RCSSWA, the calculation time is given by:

$$T_{rc} = \sum_j T^j_S + T^j_{\hat{v}\Delta}, \tag{5.23}$$

where T^j_S is the overhead time for iteration j while $T^j_{\hat{v}\Delta}$ is the time taken to calculate the Newton update using Equation (5.3). The RCSSWA overhead time is equal to the amount of time taken to find S^j_Δ and test $\|\hat{\boldsymbol{d}}_\Delta\|_2$ on iterations where the conditions in Equation (5.21) are met. On iterations where the conditions in Equation (5.21) are not met, it is equal to the total time taken to find S^j_Δ, test $\|\hat{\boldsymbol{d}}_\Delta\|_2$, and calculate $\Delta\hat{\boldsymbol{v}}_g$.

5.2.1.1 Inverter Chain

The first case study is the CMOS inverter chain shown in Figure 4.1. The input and output waveforms for the inverter chain with both RCSSWA and SSWA are shown in Figure 5.1. There is no significant difference in the final result between SSWA and RCSSWA.

The simulation run times, calculation times, RCSSWA overhead times and number of iterations are shown in Table 5.1. There is a difference in the number of iterations between SSWA and RCSSWA that is due to the difference between $\Delta\hat{\boldsymbol{v}}^{j+1}$ and $\Delta\hat{\boldsymbol{v}}^{j+1}_\Delta$. The RCSSWA overhead time accounts for only a fraction of the total calculation time and, therefore, the overhead of performing the support search and refinement are not a dominant factor with this circuit. There are slowdowns seen with the Coif2 and Sym3 wavelets. The per-iteration simulation and calculation times are lower for RCSSWA with these wavelets which indicates that the slowdowns are due to the increased number of iterations of RCSSWA. The SSTD simulation time was 30.6615 seconds and took 31 iterations (0.9891 seconds per-iteration). None of the wavelet simulations ran faster than the time domain nor did they have lower calculated per-iteration simulation times.

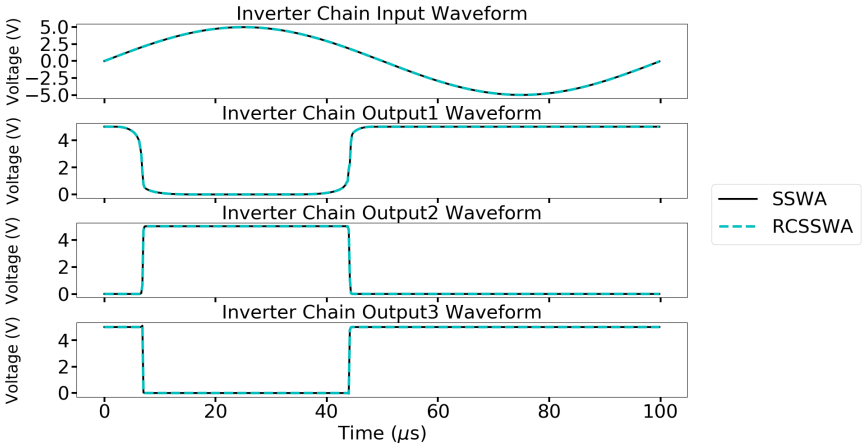

Figure 5.1: Input and output waveforms for the inverter chain. The solid line represents the SSWA simulation results and the dashed line represents the RCSSWA simulation results.

Table 5.1: Total simulation, calculation, RCSSWA overhead, symbolic factorization, and numeric factorization timing data and number of iterations for the inverter chain. Green font indicates where the RCSSWA method is faster than the SSWA method.

	Haar	Db2	Db4	Coif2	Sym2	Sym3	Bior4.4	Bior6.8
SSWA Sim Time	43.35 [1.204]	48.48 [1.310]	53.87 [1.738]	68.35 [1.667]	43.97 [1.293]	60.98 [1.418]	57.47 [1.596]	85.85 [2.094]
RCSSWA Sim Time	40.69 [1.100]	43.95 [1.157]	48.88 [1.397]	70.73 [1.505]	38.09 [1.190]	66.67 [1.361]	53.59 [1.410]	67.67 [1.440]
Speedup	1.065	1.103	1.102	0.966	1.154	0.915	1.072	1.269
SSWA Symbolic Time	0.295 [0.008]	0.391 [0.011]	0.499 [0.016]	0.739 [0.018]	0.372 [0.011]	0.528 [0.012]	0.540 [0.015]	0.822 [0.020]
RCSSWA Symbolic Time	0.295 [0.008]	0.422 [0.011]	0.611 [0.018]	0.856 [0.018]	0.379 [0.012]	0.658 [0.013]	0.574 [0.015]	0.755 [0.016]
SSWA Numeric Time	0.926 [0.026]	1.812 [0.049]	9.885 [0.319]	10.72 [0.262]	1.686 [0.050]	3.921 [0.091]	5.935 [0.165]	12.61 [0.308]
RCSSWA Numeric Time	1.207 [0.033]	1.926 [0.051]	4.817 [0.138]	6.889 [0.147]	1.984 [0.062]	4.079 [0.083]	5.037 [0.133]	6.584 [0.140]
SSWA Calculation Time	5.153 [0.143]	6.895 [0.186]	16.00 [0.516]	19.784 [0.483]	6.367 [0.187]	10.751 [0.250]	12.466 [0.346]	23.515 [0.574]
RCSSWA Calculation Time	4.280 [0.116]	5.092 [0.134]	8.744 [0.250]	12.600 [0.268]	4.830 [0.151]	8.725 [0.178]	9.270 [0.244]	11.95 [0.254]
SSWA Iterations	36	37	31	41	34	43	36	41
RCSSWA Iterations	37	38	35	47	32	49	38	47
RCSSWA Overhead Time	0.127 [0.003]	0.473 [0.013]	0.558 [0.016]	1.228 [0.026]	0.345 [0.011]	0.679 [0.014]	0.998 [0.026]	0.881 [0.019]
Note: Calculated per-iteration values shown in square brackets and all times are in seconds.								

The percentage of non-zero entries in S_Δ^j and the average number of non-zero elements in the Jacobian

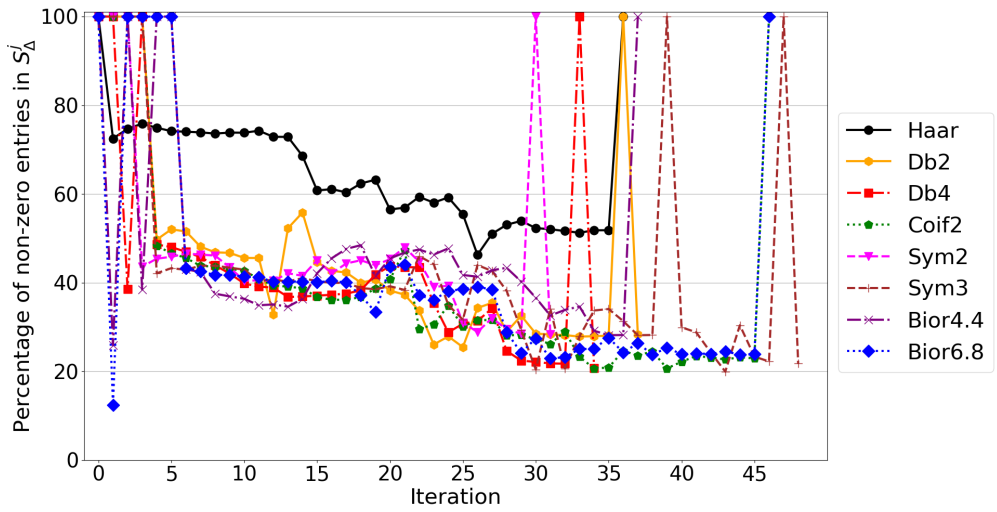

Figure 5.2: Percentage of non-zero entries in S_Δ^j for each iteration for the inverter chain.

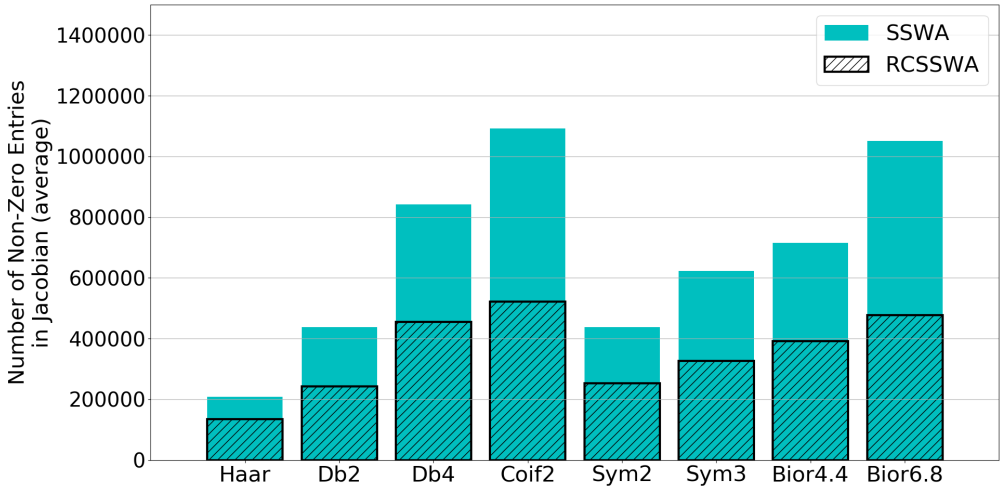

Figure 5.3: Average number of non-zero entries in the Jacobian matrices for the inverter chain.

matrices are shown in Figures 5.2 and 5.3 respectively. The Haar wavelet simulations exhibited the lowest per-iteration calculation times. This is expected since the number of non-zero elements in the Jacobian matrices of the Haar wavelet are lower than all other wavelets with both SSWA and RCSSWA. All iterations where the percentage of non-zero entries in S_Δ^j is 100%, excluding iteration 0, are iterations where the conditions in Equation (5.21) are not met. From Equation (5.21), $\|\hat{\boldsymbol{d}}_\Delta\|_2$ has to be less than $\alpha \cdot \|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\|_2$. As the simulation approaches convergence, $\|\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\|_2$ decreases and $\|\hat{\boldsymbol{d}}_\Delta\|_2$ has a higher chance of being high enough for the conditions in Equation (5.21) to not be met and causes the later iterations to use $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$ instead of $\hat{\boldsymbol{J}}_\Delta$. Since the majority of iterations with RCSSWA utilized $\hat{\boldsymbol{J}}_\Delta$, the overhead times are very low and there is a reduction in calculation times with RCSSWA.

The average density of the L_1, U, and L_2 matrices are shown in Figure 5.4. The density of the L_1

and U matrices follow a similar pattern as the Jacobian densities (see Figure 4.3) with the Haar, Db2, and Sym2 wavelets exhibiting the lowest densities. The L_2 matrix exhibits the highest average densities.

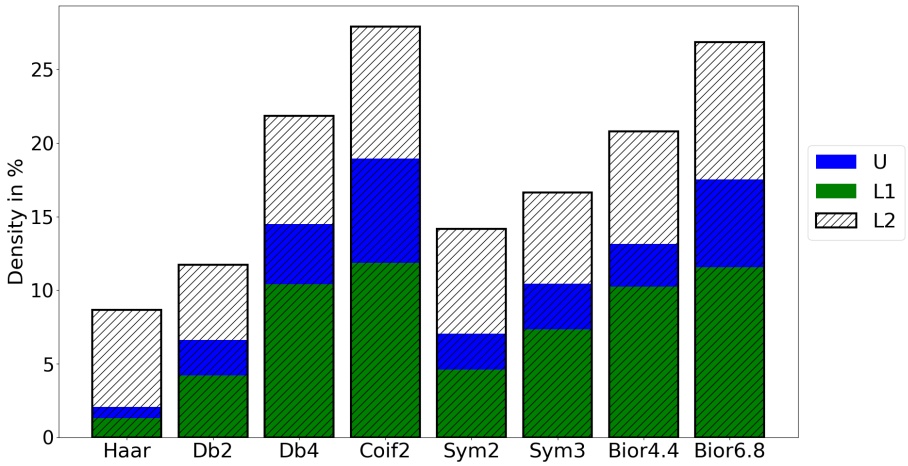

Figure 5.4: Average density of the L_1, U, and L_2 matrices for the inverter chain circuit.

5.2.1.2 Gilbert Cell

The second case study is the Gilbert cell circuit shown in Figure 4.7. The input and output waveforms for the Gilbert cell with both SSWA and RCSSWA are shown in Figure 5.5. As is the case with the inverter chain, there is no significant difference between the final result for SSWA and RCSSWA.

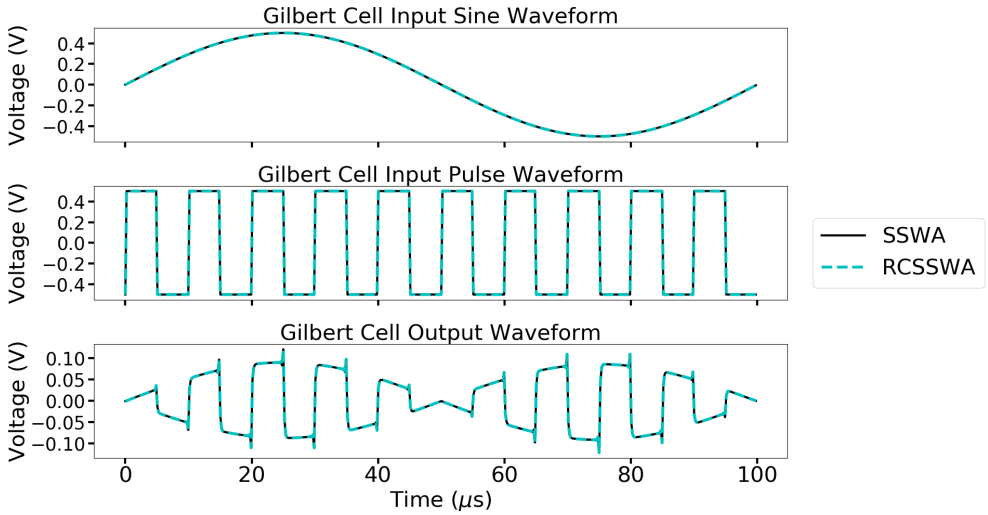

Figure 5.5: Input and output waves for the Gilbert cell. The solid line represents the SSWA simulation results and the dashed line represents the RCSSWA simulation results.

Table 5.2: Total simulation, calculation, RCSSWA overhead, symbolic factorization, and numeric factorization timing data and number of iterations for the Gilbert cell. Green font indicates where the RCSSWA method is faster than the SSWA method.

	Haar	Db2	Db4	Coif2	Sym2	Sym3	Bior4.4	Bior6.8
SSWA Sim Time	17.60 [2.514]	25.06 [3.581]	47.05 [6.722]	53.98 [7.712]	24.74 [3.535]	42.32 [6.046]	48.56 [6.937]	54.84 [7.834]
RCSSWA Sim Time	20.39 [2.548]	30.23 [3.778]	61.25 [7.657]	59.44 [8.492]	30.55 [3.819]	44.25 [6.321]	63.45 [7.931]	65.51 [8.189]
Speedup	0.863	0.829	0.768	0.908	0.810	0.957	0.765	0.837
SSWA Symbolic Time	0.797 [0.114]	0.870 [0.124]	0.985 [0.141]	1.151 [0.164]	0.853 [0.122]	0.928 [0.133]	0.924 [0.132]	1.133 [0.162]
RCSSWA Symbolic Time	0.844 [0.106]	1.000 [0.125]	1.355 [0.169]	1.373 [0.196]	1.010 [0.126]	1.043 [0.149]	1.231 [0.154]	1.439 [0.180]
SSWA Numeric Time	5.021 [0.717]	9.6901 [1.3843]	26.9405 [3.8486]	31.8360 [4.5480]	9.8344 [1.4049]	23.9231 [3.4176]	29.0176 [4.1454]	32.5335 [4.6476]
RCSSWA Numeric Time	6.838 [0.855]	13.4601 [1.6825]	36.9160 [4.6145]	36.5964 [5.2281]	13.5655 [1.6957]	26.0617 [3.7231]	40.7488 [5.0936]	39.4250 [4.9281]
SSWA Calculation Time	8.572 [1.225]	14.80 [2.115]	35.20 [5.028]	40.90 [5.843]	14.86 [2.123]	31.35 [4.479]	37.11 [5.302]	41.75 [5.965]
RCSSWA Calculation Time	10.79 [1.349]	19.112 [2.389]	46.95 [5.869]	45.79 [6.542]	19.28 [2.410]	33.36 [4.765]	50.88 [6.360]	50.94 [6.368]
SSWA Iterations	7	7	7	7	7	7	7	7
RCSSWA Iterations	8	8	8	7	8	7	8	8
RCSSWA Overhead Time	1.373 [0.172]	4.246 [0.531]	10.85 [1.357]	8.455 [1.208]	4.359 [0.545]	7.148 [1.021]	12.52 [1.565]	9.287 [1.161]
Note: Calculated per-iteration values shown in square brackets and all times are in seconds.								

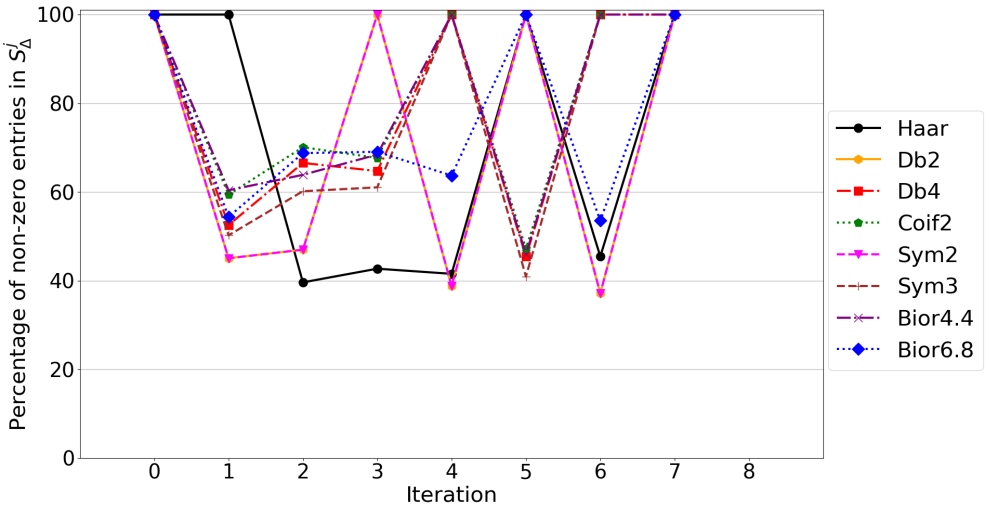

Figure 5.6: Percentage of non-zero entries in S_Δ^j for each iteration for the Gilbert cell.

The simulation run times, calculation times, RCSSWA overhead times, and number of iterations are

shown in Table 5.2. None of the RCSSWA simulations ran faster than the SSWA simulations. The per-iteration calculation times are higher with RCSSWA and the per-iteration RCSSWA overhead times are high enough to account for the slowdowns seen with RCSSWA. The SSTD simulation time was 13.4959 seconds and took 8 iterations (1.6870 seconds per-iteration). None of the RCSSWA simulations had lower simulation times than SSTD.

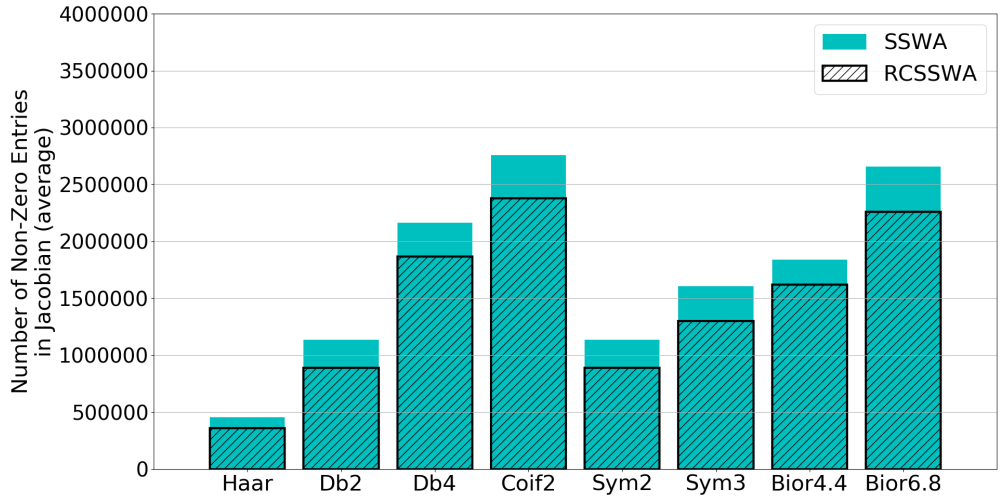

Figure 5.7: Average number of non-zero entries in the Jacobian matrices for the Gilbert cell.

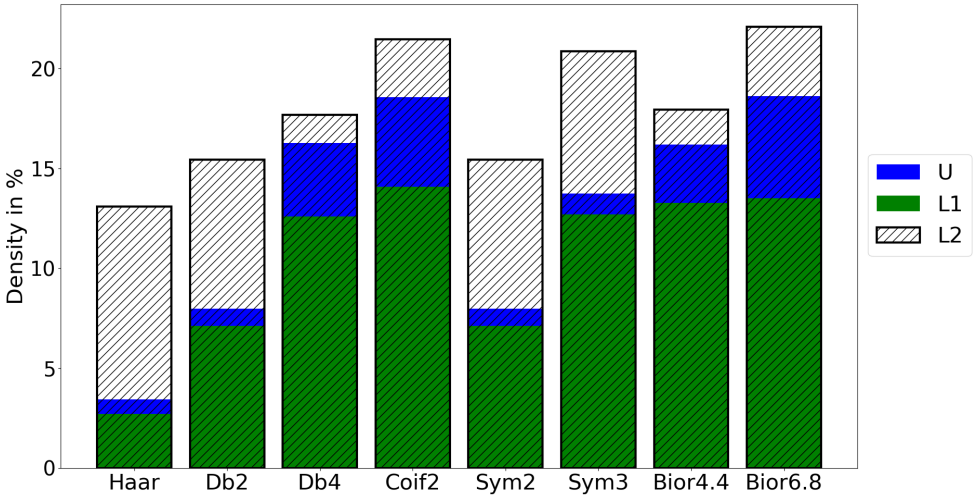

Figure 5.8: Average density of the L_1, U, and L_2 matrices for the Gilbert cell circuit.

The percentage of non-zero entries in S_Δ^j and the average number of non-zero elements in the Jacobian matrices for the Gilbert cell are shown in Figures 5.6 and 5.7 respectively. All iterations where the percentage of non-zero entries in S_Δ^j is 100%, excluding iteration 0, are iterations where the conditions in Equation (5.21) are not met. The lowest calculation times occur with the Haar wavelet which has the lowest average number of non-zero entries in the Jacobian matrix with both SSWA and RCSSWA. The

reduction in average number of non-zero elements is not as high as with the inverter chain circuit.

The average density of the L_1, U, and L_2 matrices are shown in Figure 5.8. The density of the L_1 and U matrices follow a similar pattern as the Jacobian densities (see Figure 4.9) with the Haar, Db2, and Sym2 wavelets exhibiting the lowest densities. The L_2 matrix exhibits the highest average densities for all test cases with the U matrix being the second most dense.

5.2.1.3 Ultrasound Transducer Driver

The third case study is the ultrasound transducer driver circuit shown in Figure 4.13. The input and output waveforms for the transducer driver circuit with both SSWA and RCSSWA are shown in Figure 5.9. As with the inverter chain and Gilbert cell, there is no significant difference between the SSWA and RCSSWA nodal variable waveforms.

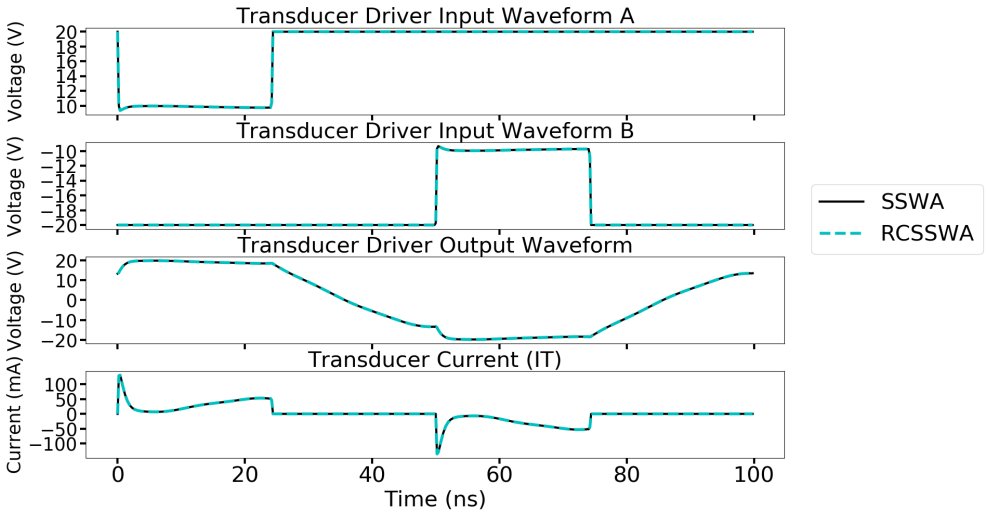

Figure 5.9: Waveforms for the transducer driver circuit. The solid line represents the SSWA simulation results and the dashed line represents the RCSSWA simulation results.

The simulation run times, calculation times, RCSSWA overhead times, and number of iterations for the transducer driver are shown in Table 5.3. There are speedups with all but the Haar wavelet. Based on per-iteration times, the slowdown with the Haar wavelet is due to both the extra overhead associated with RCSSWA as well as the increase in number of iterations. The number of iterations is altered by RCSSWA with all of the test cases and is high enough to almost cancel out the speedup in per-iteration times with the Sym2, Sym3, and Bior4.4 wavelets. As is the case with the Gilbert cell, the RCSSWA overhead times account for a fraction of the calculation times. The SSTD simulation time was 11.7099 seconds and took 11 iterations (1.0645 seconds per-iteration). Based on per-iteration times, the speedups seen with RCSSWA are not enough to lead to faster simulation times than SSTD.

Table 5.3: Total simulation, calculation, RCSSWA overhead, symbolic factorization, and numeric factorization timing data and number of iterations for the transducer driver. Green font indicates where the RCSSWA method is faster than the SSWA method.

	Haar	Db2	Db4	Coif2	Sym2	Sym3	Bior4.4	Bior6.8
SSWA Sim Time	21.38 [2.138]	17.59 [1.955]	24.56 [2.729]	34.93 [4.366]	16.76 [1.862]	21.67 [2.407]	29.79 [3.724]	38.80 [4.312]
RCSSWA Sim Time	26.54 [2.212]	15.07 [1.507]	20.12 [1.829]	28.77 [2.397]	15.29 [1.390]	19.94 [1.813]	27.47 [1.831]	25.71 [2.338]
Speedup	0.806	1.168	1.221	1.214	1.096	1.086	1.085	1.509
SSWA Symbolic Time	0.329 [0.033]	0.415 [0.046]	0.530 [0.059]	0.492 [0.062]	0.406 [0.045]	0.439 [0.049]	0.451 [0.056]	0.603 [0.067]
RCSSWA Symbolic Time	0.419 [0.035]	0.373 [0.037]	0.465 [0.042]	0.516 [0.043]	0.399 [0.036]	0.452 [0.041]	0.559 [0.037]	0.465 [0.042]
SSWA Numeric Time	16.04 [1.605]	11.56 [1.284]	17.66 [1.962]	28.02 [3.503]	10.89 [1.211]	15.35 [1.706]	23.45 [2.931]	31.05 [3.450]
RCSSWA Numeric Time	20.20 [1.683]	8.986 [0.899]	12.72 [1.156]	20.48 [1.706]	8.875 [0.807]	12.91 [1.174]	18.48 [1.232]	18.02 [1.638]
SSWA Calculation Time	18.26 [1.826]	14.40 [1.600]	21.05 [2.339]	31.67 [3.959]	13.55 [1.506]	18.45 [2.050]	26.87 [3.359]	35.02 [3.892]
RCSSWA Calculation Time	22.59 [1.882]	11.24 [1.124]	15.30 [1.391]	23.77 [1.981]	11.22 [1.020]	15.75 [1.432]	21.99 [1.466]	20.81 [1.892]
SSWA Iterations	10	9	9	8	9	9	8	9
RCSSWA Iterations	12	10	11	12	11	11	15	11
RCSSWA Overhead Time	1.331 [0.111]	0.682 [0.068]	0.875 [0.080]	1.165 [0.097]	0.645 [0.059]	0.936 [0.085]	1.591 [0.106]	1.276 [0.116]
Note: Calculated per-iteration values shown in square brackets and all times are in seconds.								

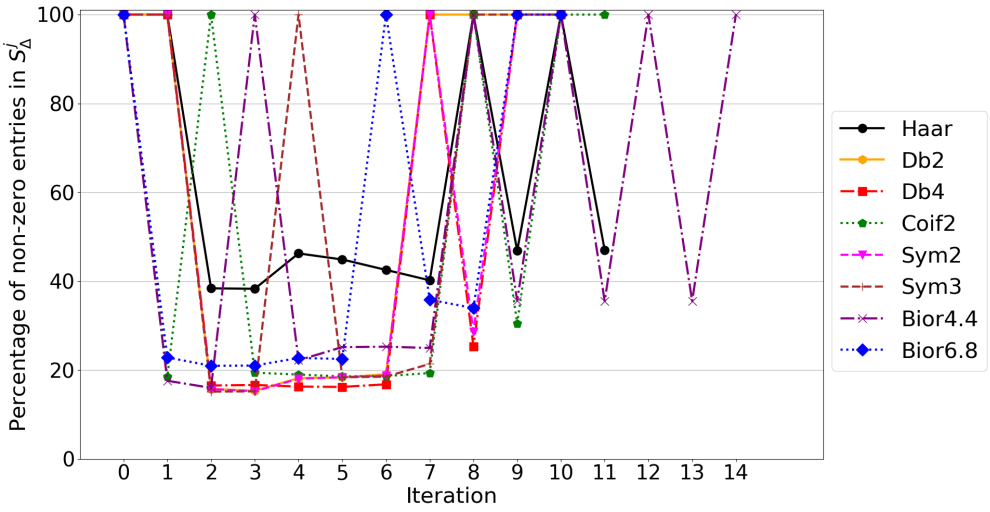

Figure 5.10: Percentage of non-zero entries in S_Δ^j for each iteration for the transducer driver.

The percentage of of non-zero entries in S_Δ^j and the average number of non-zero elements in the Jacobian matrices for the transducer driver are shown in Figures 5.10 and 5.11 respectively. All iterations where the percentage of non-zero entries in S_Δ^j is 100%, excluding iteration 0, are iterations where the conditions in Equation (5.21) are not met. There is a very small reduction in Jacobian matrix density with the Haar wavelet and the RCSSWA per-iteration calculation time is roughly the same as SSWA. Therefore, the increase in speed from RCSSWA with the Haar wavelet is overcome by the calculation overhead of RCSSWA.

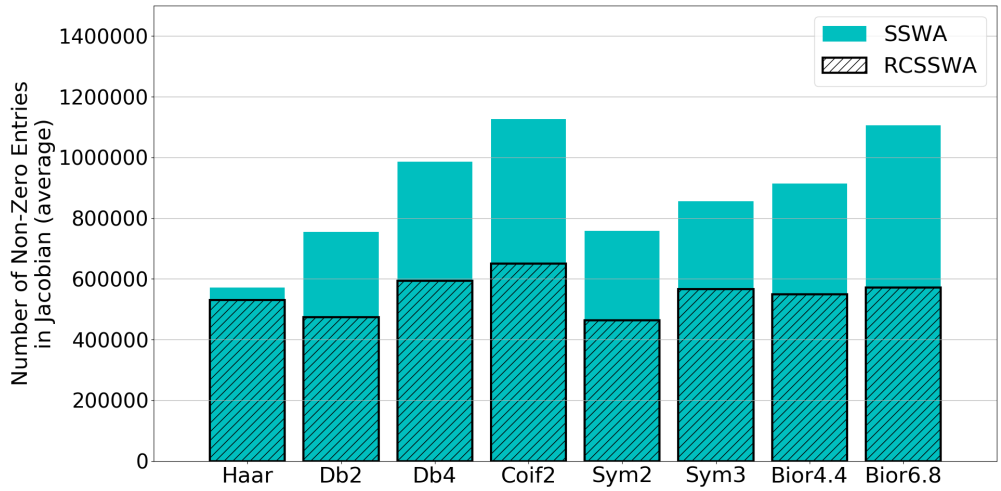

Figure 5.11: Average number of non-zero entries in the Jacobian matrices for the transducer driver.

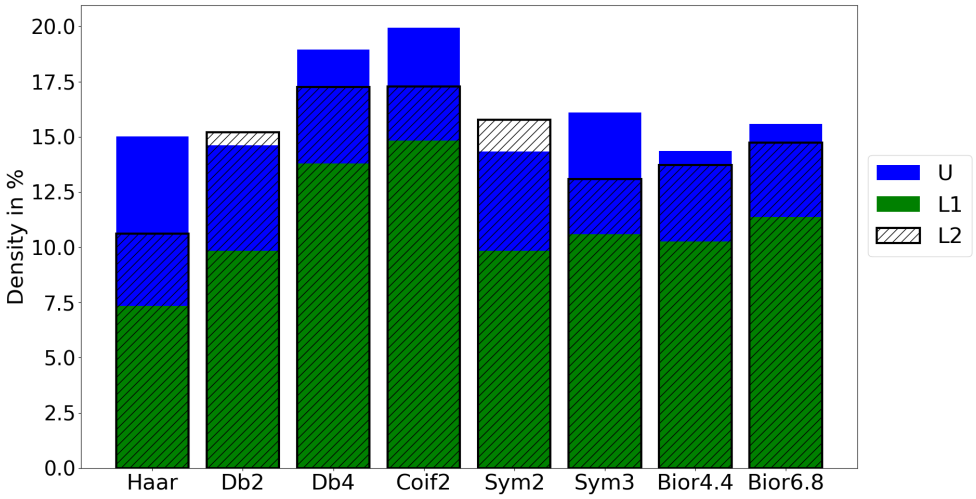

Figure 5.12: Average density of the L_1, U, and L_2 matrices for the ultrasound transducer driver circuit.

The average density of the L_1, U, and L_2 matrices are shown in Figure 5.12. The density of the U matrices exhibit, on average, higher densities than the L_2 matrix with most wavelets.

5.2.1.4 Transmission Line Circuit

The fourth case study is the transmission line circuit shown in Figure 4.21. The non-thresholded (solid line) and thresholded (dashed line) input and output waveforms and the voltages at two of the transmission line ports with both SSWA and RCSSWA are shown in Figure 5.13. As with the other circuits in this study, there is no significant difference between the SSWA and RCSSWA nodal variable waveforms.

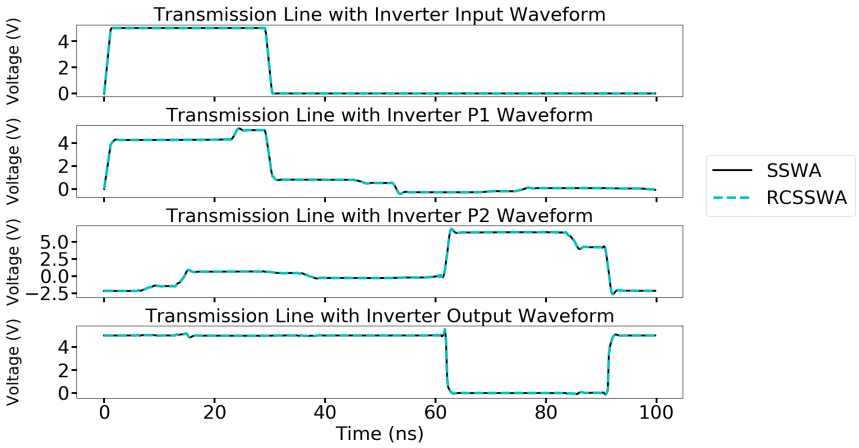

Figure 5.13: Input and output nodal variables for the transmission line circuit. The solid line represents the SSWA simulation results and the dashed line represents the RCSSWA simulation results.

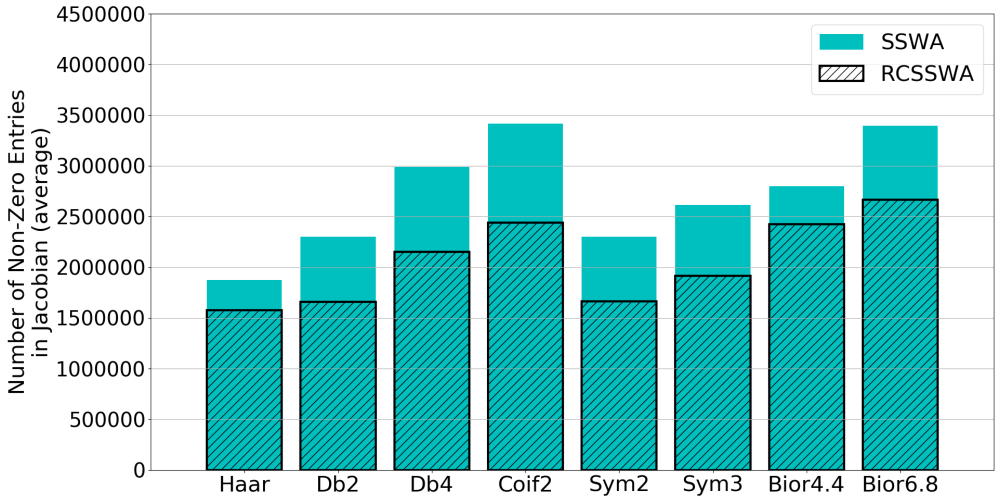

Figure 5.14: Average number of non-zero entries in the Jacobian matrices for the transmission line circuit.

The simulation run times, calculation times, RCSSWA overhead times, and number of iterations for the transmission line circuit are shown in Table 5.4. There are very slight speedups over SSWA with most of the test wavelets. Based on per-iteration simulation and calculation times, the speedup with the Coif2 wavelet is due to the decrease in number of iterations. The per-iteration RCSSWA overhead times

indicate that the RCSSWA calculation overhead is the cause of the slowdowns in calculation times seen with the Haar, Coif2, Bior4.4, and Bior6.8 wavelets. The SSTD simulation time was 86.4077 seconds and took 34 iterations (2.5414 seconds per-iteration). The speedup with the Db2 wavelet caused the RCSSWA method to exhibit a speedup over SSTD. As is the case with the other circuits, the RCSSWA overhead times account for a fraction of the RCSSWA calculation time.

The percentage of of non-zero entries in S_{Δ}^{j} and the average number of non-zero elements in the Jacobian matrices for the transmission line circuit are shown in Figures 5.15 and 5.14 respectively. All iterations where the percentage of non-zero entries in S_{Δ}^{j} is 100%, excluding iteration 0, are iterations where the conditions in Equation (5.21) are not met.

The average density of the L_1, U, and L_2 matrices are shown in Figure 5.16. The density of the L_1 and U matrices follow a similar pattern as the Jacobian densities (see Figure 4.23). The L_2 matrix exhibits the highest average densities.

Table 5.4: Total simulation, calculation, RCSSWA overhead, symbolic factorization, and numeric factorization timing data and number of iterations for the transmission line circuit. Green font indicates where the RCSSWA method is faster than the SSWA method.

	Haar	Db2	Db4	Coif2	Sym2	Sym3	Bior4.4	Bior6.8
SSWA Sim Time	98.97 [2.828]	86.89 [2.633]	127.9 [3.553]	159.2 [4.191]	89.77 [2.720]	101.3 [3.068]	126.5 [3.720]	154.1 [4.403]
RCSSWA Sim Time	104.0 [2.972]	84.75 [2.493]	126.4 [3.511]	152.4 [4.355]	86.428 [2.542]	100.3 [3.039]	155.7 [4.098]	174.3 [4.840]
Speedup	0.951	1.025	1.012	1.045	1.039	1.010	0.812	0.884
SSWA Symbolic Time	3.069 [0.088]	4.183 [0.127]	7.099 [0.197]	8.611 [0.227]	4.465 [0.135]	4.822 [0.146]	7.203 [0.212]	9.501 [0.272]
RCSSWA Symbolic Time	3.385 [0.097]	3.328 [0.098]	5.081 [0.141]	5.927 [0.169]	3.468 [0.102]	4.154 [0.126]	8.088 [0.213]	7.154 [0.199]
SSWA Numeric Time	59.55 [1.701]	43.40 [1.315]	63.56 [1.766]	84.32 [2.219]	43.98 [1.333]	52.14 [1.580]	65.718 [1.933]	80.89 [2.311]
RCSSWA Numeric Time	67.29 [1.926]	43.58 [1.282]	68.98 [1.916]	87.76 [2.508]	44.73 [1.316]	53.25 [1.614]	88.660 [2.333]	102.3 [2.842]
SSWA Calculation Time	80.09 [2.288]	66.25 [2.007]	99.44 [2.762]	125.4 [3.299]	68.28 [2.07]	78.78 [2.387]	99.25 [2.919]	122.3 [3.493]
RCSSWA Calculation Time	84.97 [2.428]	61.21 [1.800]	94.378 [2.622]	117.2 [3.348]	62.74 [1.85]	74.35 [2.253]	125.2 [3.294]	137.9 [3.830]
SSWA Iterations	35	33	36	38	33	33	34	35
RCSSWA Iterations	35	34	36	35	34	33	38	36
RCSSWA Overhead Time	14.44 [0.413]	4.559 [0.134]	7.198 [0.200]	13.07 [0.373]	4.859 [0.143]	6.080 [0.184]	19.32 [0.508]	15.19 [0.422]

Note: Calculated per-iteration values shown in square brackets and all times are in seconds.

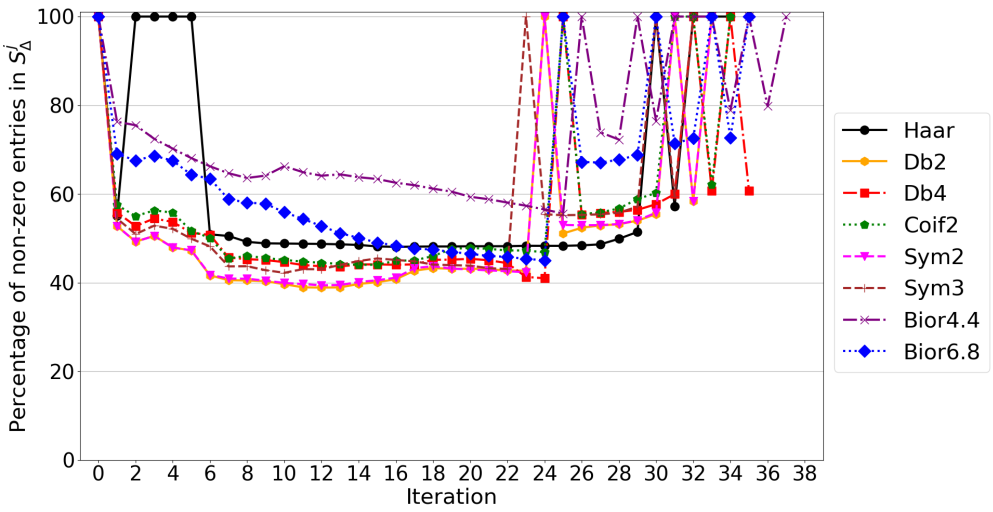

Figure 5.15: Percentage of non-zero entries in S_Δ^j for each iteration for the transmission line circuit.

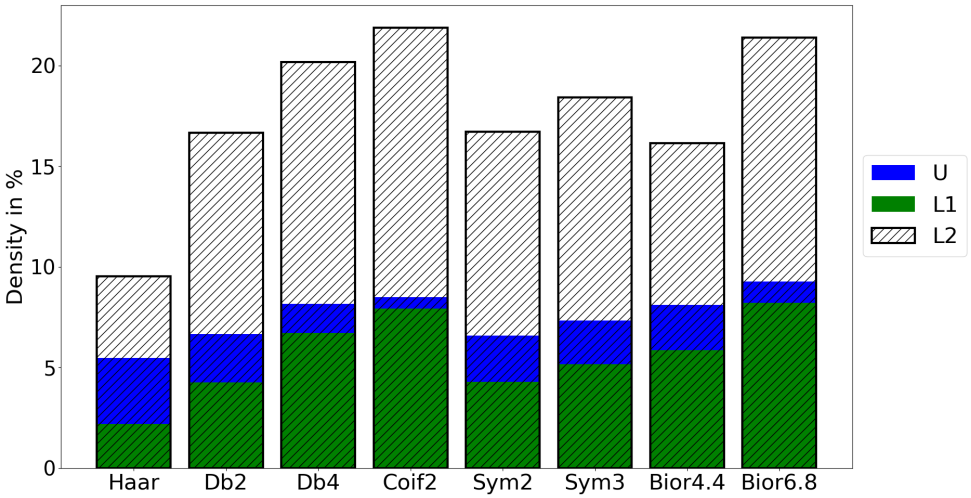

Figure 5.16: Average density of the L_1, U, and L_2 matrices for the transmission line circuit.

5.2.1.5 Stepped Impedance Filter Circuit

The last case study is the stepped impedance filter circuit shown in Figure 4.27. The non-thresholded (solid line) and thresholded (dashed line) input, output of the filter, and output of the amplifier waveforms with both SSWA and RCSSWA are shown in Figure 5.17. As with the other circuits in this study, there is no significant difference between the SSWA and RCSSWA nodal variable waveforms.

The simulation run times, calculation times, RCSSWA overhead times, and number of iterations for the stepped impedance filter circuit are shown in Table 5.5. The SSTD simulation time was 287.4568 seconds and took 6 iterations (47.9095 seconds per-iteration). With RCSSWA, both the Db2 and Bior6.8 wavelet simulations were faster than SSTD.

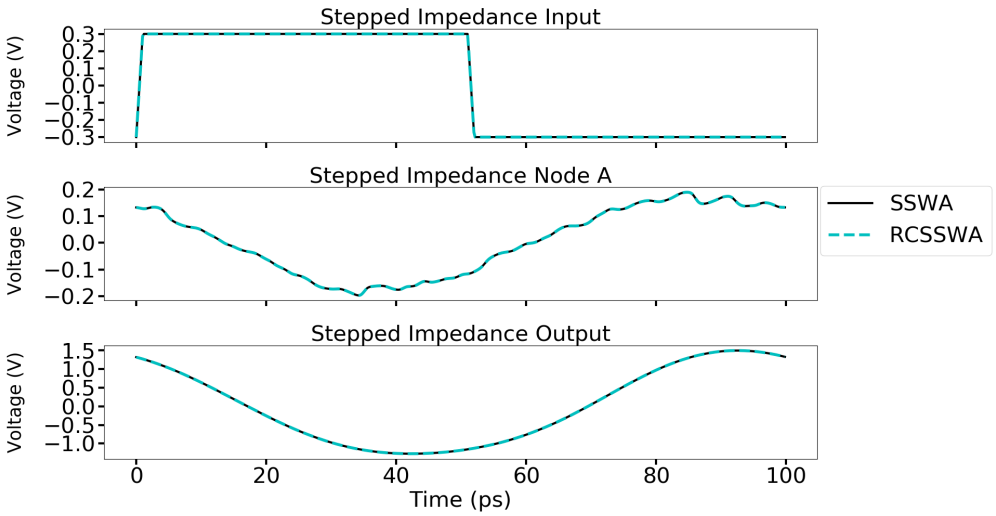

Figure 5.17: Input and output nodal variables for the stepped impedance filter circuit. The solid line represents the SSWA simulation results and the dashed line represents the RCSSWA simulation results.

Table 5.5: Total simulation, calculation, RCSSWA overhead, symbolic factorization, and numeric factorization timing data and number of iterations for the stepped impedance filter circuit. Green font indicates where the RCSSWA method is faster than the SSWA method.

	Haar	Db2	Db4	Coif2	Sym2	Sym3	Bior4.4	Bior6.8
SSWA Sim Time	328.1 [54.68]	256.2 [42.70]	267.6 [44.61]	280.3 [46.72]	246.5 [41.09]	262.4 [43.73]	289.7 [48.28]	342.0 [57.00]
RCSSWA Sim Time	533.8 [88.96]	283.3 [56.66]	317.4 [52.90]	395.5 [65.91]	338.0 [56.34]	332.6 [55.43]	419.4 [69.91]	262.1 [37.43]
Speedup	0.615	0.904	0.843	0.709	0.729	0.789	0.691	1.305
SSWA Symbolic Time	5.789 [0.965]	9.553 [1.592]	12.71 [2.119]	14.90 [2.483]	9.219 [1.537]	10.74 [1.790]	13.22 [2.203]	18.96 [3.160]
RCSSWA Symbolic Time	10.03 [1.671]	10.66 [2.132]	15.41 [2.569]	21.54 [3.590]	12.21 [2.036]	12.77 [2.128]	18.90 [3.150]	10.96 [1.566]
SSWA Numeric Time	279.6 [46.60]	190.4 [31.74]	193.4 [32.24]	206.0 [34.33]	177.5 [29.58]	192.9 [32.14]	212.7 [35.45]	251.3 [41.88]
RCSSWA Numeric Time	454.3 [75.72]	211.0 [42.20]	233.2 [38.87]	286.0 [47.67]	251.1 [41.85]	241.1 [40.18]	311.9 [51.99]	185.2 [26.46]
SSWA Calculation Time	320.8 [53.47]	248.5 [41.41]	259.1 [43.18]	271.7 [45.28]	239.3 [39.89]	254.3 [42.39]	280.8 [46.81]	331.3 [55.21]
RCSSWA Calculation Time	523.2 [87.20]	274.2 [54.84]	305.7 [50.95]	381.9 [63.65]	326.9 [54.48]	321.6 [53.59]	407.8 [67.96]	248.4 [35.48]
SSWA Iterations	6	6	6	6	6	6	6	6
RCSSWA Iterations	6	5	6	6	6	6	6	7
RCSSWA Overhead Time	194.9 [32.48]	75.64 [15.13]	62.69 [10.45]	88.03 [14.67]	92.31 [15.38]	74.49 [12.42]	118.6 [19.76]	33.64 [4.805]
Note: Calculated per-iteration values shown in square brackets and all times are in seconds.								

However, the Db2 wavelet simulation would have run slower than SSTD if the number of iterations remained unchanged. The Haar, Coif2, and Bior4.4 wavelets had no iterations where the the conditions in Equation (5.21) were met. Thus, RCSSWA does not provide a benefit with these wavelets on this circuit. The per-iteration RCSSWA overhead time is high enough to account for the slowdown with the Db2, Db4, Sym2, and Sym3 wavelets. The majority of iterations requires support refinement with these wavelets and, therefore, the overhead of implementing RCSSWA overcomes any increase in speed that occurs with these wavelets.

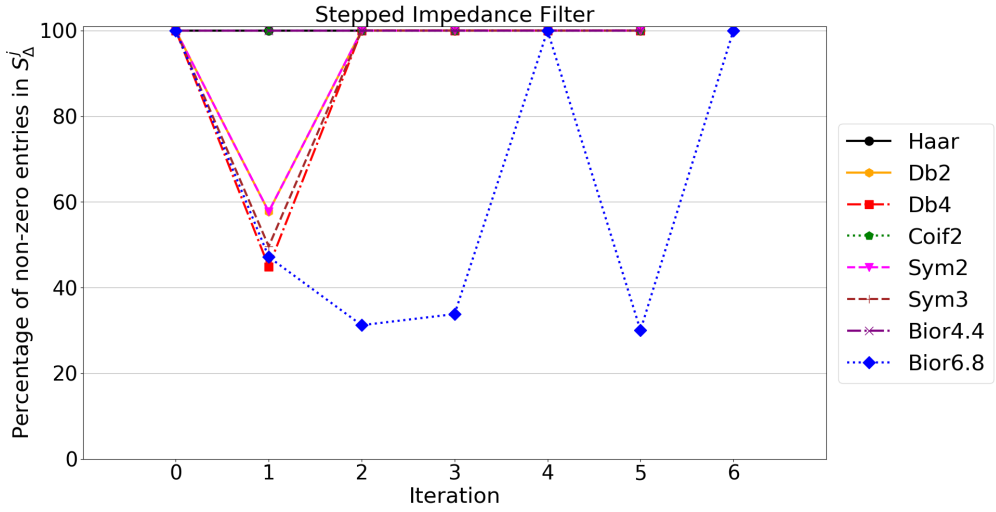

Figure 5.18: Percentage of non-zero entries in S_Δ^j for each iteration for the stepped impedance filter circuit.

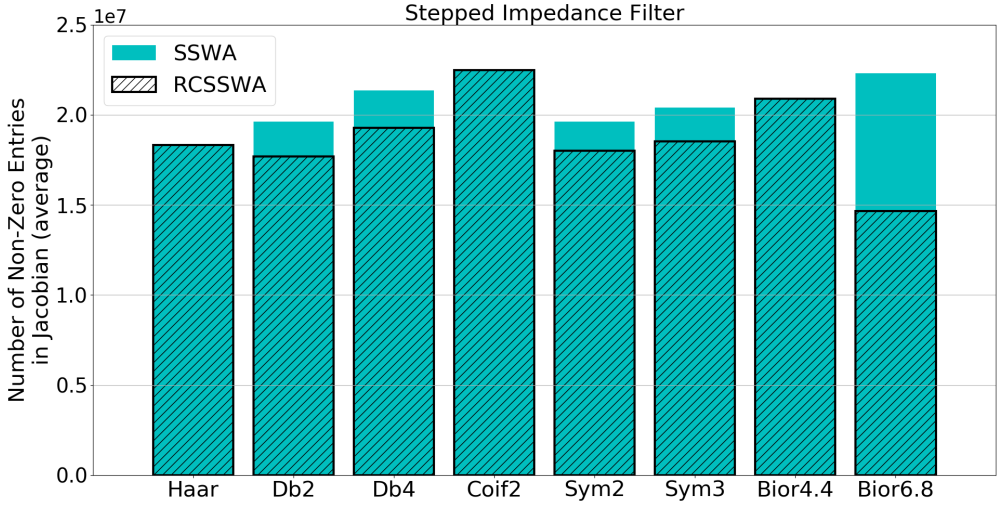

Figure 5.19: Average number of non-zero entries in the Jacobian matrices for the stepped impedance filter circuit.

The percentage of of non-zero entries in S_Δ^j and the average number of non-zero elements in the

Jacobian matrices for the stepped impedance filter circuit are shown in Figures 5.18 and 5.19 respectively. All iterations where the percentage of non-zero entries in S_Δ^j is 100%, excluding iteration 0, are iterations where the conditions in Equation (5.21) are not met. There was no reduction in percentage of non-zero entries in S_Δ^j for the Haar, Coif2, and Bior4.4 wavelets. The support refinement was applied for the majority of iterations in all cases but the Bior6.8 wavelet which caused the RCSSWA overhead times to have a much higher effect on the RCSSWA calculation times with this circuit than the others.

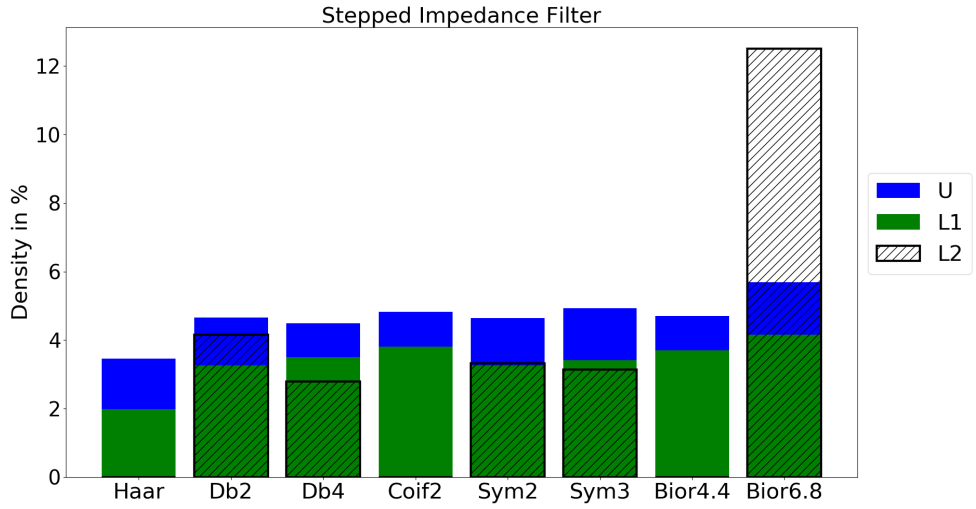

Figure 5.20: Average density of the L_1, U, and L_2 matrices for the stepped impedance filter circuit.

The average density of the L_1, U, and L_2 matrices are shown in Figure 5.20. The density of the L_1 and U matrices do not follow a similar pattern as the Jacobian densities (see Figure 4.29). The L_2 matrix does not always exhibit the highest average densities. For the Haar, Coif2, and Bior4.4 wavelets, the $\hat{\boldsymbol{J}}_\Delta$ matrix was never utilized so there is no L_2 matrix density reported for them.

5.2.2 Discussion

RCSSWA lead to a speedup in per-iteration calculation time over SSWA, and even SSTD, in some of the test cases. This speedup tended to be low but was significant in a few of the test cases. The change in number of iterations with RCSSWA indicates that there is a significant deviation between $\Delta\hat{\boldsymbol{v}}^{j+1}$ and $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$ when using RCSSWA. This deviation is caused by errors in the support estimate and can lead to enough extra iterations to cause a slowdown in total simulation time.

One of the disadvantages of using RCSSWA is the overhead time introduced in estimating and refining S_Δ^j. In cases where the size of $\hat{\boldsymbol{J}}_\Delta$ is reduced significantly enough, the overhead time of RCSSWA does not have a large effect on the per-iteration calculation time. However, in cases where there is no reduction for a majority of iterations, as is the case with the stepped impedance filter, the overhead time can be significant. There are wavelets that provide a speed/sparsity advantage over others with each of

the example circuits. However, based on this data, there is no wavelet that will provide the sparsest representation, and fastest performance, in all cases. One could select a wavelet based on the expected final result of the nodal variable waveforms but, since the nodal variable waveforms change as the Newton updates are calculated, this does not guarantee that the sparsest representations for all iterations. Thus, it is difficult to select an appropriate wavelet for use with all iterations in advance with RCSSWA.

The L_2 matrix can exhibit a much higher average density then the L_1 and U matrices but there appears to be no way to predict for which circuit problems the L_2 matrix will exhibit the greatest densities. This could cause issues with larger circuits since the density of the L_2 matrix has the potential to be much higher than the densities of the L_1 and U matrices. The Haar wavelet tended to exhibit the lowest densities of all the cases tested and appears to be the best wavelet to utilize to achieve the lowest density Jacobian, L_1, U, and L_2 matrices if the optimum wavelet is not known.

5.3 Hybrid Reduced Column Steady-State Wavelet Analysis

The main disadvantage of the SSWA formulations used in this thesis is the computational overhead due to the wavelet transforms involved with converting the system of equations into the wavelet domain. As a result, SSWA, SSWA with adaptive Jacobian matrix thresholding, and RCSSWA all suffer a computational disadvantage compared to SSTD. However, with a small alteration to the formulation of Equation (2.58), it is possible to utilize RCSSWA to create a hybrid wavelet-time approach for the Newton updates.

Substituting $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j) = \boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}^{-1}$ and $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) = \boldsymbol{W}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j)$ into Equation (2.58) yields:

$$\boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}^{-1}\Delta\hat{\boldsymbol{v}}^{j+1} = -\boldsymbol{W}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j). \tag{5.24}$$

Equation (5.24) can be partitioned into elements that are indexed by set S_Δ^{j+1} and not indexed by set S_Δ^{j+1} as:

$$\boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}\Delta\hat{\boldsymbol{v}}_\Delta^{j+1} + \boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_{\sim\Delta}^{-1}\Delta\hat{\boldsymbol{v}}_{\sim\Delta}^{j+1} = -\boldsymbol{W}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j), \tag{5.25}$$

where $\boldsymbol{W}_{\sim\Delta}^{-1}$ is formed out of the columns of $\boldsymbol{W}^{-1}$ that are not indexed by set S_Δ^{j+1}. Since all elements in $\Delta\hat{\boldsymbol{v}}^{j+1} \notin S_\Delta^{j+1}$ are expected to be equal to zero, $\boldsymbol{W}_{\sim\Delta}^{-1}\Delta\hat{\boldsymbol{v}}_{\sim\Delta}^{j+1} = \hat{\boldsymbol{0}}$ and Equation (5.25) becomes:

$$\boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}\cdot\Delta\hat{\boldsymbol{v}}_\Delta^{j+1} = -\boldsymbol{W}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j). \tag{5.26}$$

Removing the pre-multiplication by $\boldsymbol{W}$ from Equation (5.26) results in:

$$\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}\cdot(\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}) \;\; = \;\; -\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j), \tag{5.27}$$

where $\boldsymbol{W}_\Delta^{-1}$ is formed out of the columns of $\boldsymbol{W}^{-1}$ that are indexed by set S_Δ^{j+1}.

Another disadvantage of SSWA and RCSSWA is, based on the data presented in Chapter 4 and Section 5.2.1, there appears to be no one wavelet that will provide the sparsest nodal variable representations for

all cases which makes selection of an appropriate wavelet basis difficult. This problem can be addressed by the formulation in Equation (5.27) because it does not require a wavelet to be selected in advance[I]. Thus, each iteration, a wavelet that is expected to provide the sparsest representation of the nodal variable waveforms could be selected prior to forming $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}$. With the formulation in Equation (5.27), it is also possible to estimate S_v^{j+1} without using $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$:

$$\hat{\boldsymbol{g}}_v = \left(\boldsymbol{W}^{-1}\right)^T \bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)^T \cdot (-\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j) + \bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j) \cdot \bar{\boldsymbol{v}}^j). \tag{5.28}$$

With orthogonal wavelets, $\boldsymbol{W}^{-1} = \boldsymbol{W}^T$ and Equation (5.28) becomes:

$$\hat{\boldsymbol{g}}_v = \boldsymbol{W}\bar{\boldsymbol{g}}_v = \boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)^T \cdot (-\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j) + \bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j) \cdot \bar{\boldsymbol{v}}^j). \tag{5.29}$$

With biorthogonal wavelets, $\boldsymbol{W}^{-1} = \boldsymbol{W}_b^T$, and Equation (5.28) becomes:

$$\hat{\boldsymbol{g}}_v = \boldsymbol{W}_b\bar{\boldsymbol{g}}_v = \boldsymbol{W}_b\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)^T \cdot (-\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j) + \bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j) \cdot \bar{\boldsymbol{v}}^j), \tag{5.30}$$

where $\boldsymbol{W}_b = I_N \otimes W_b$. Reverse biorthogonal wavelets use Equation (5.29).

Equations (5.28), (5.29), and (5.30) have an advantage over support estimation with RCSSWA: Because $\hat{\boldsymbol{g}}_v$ does not require $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^j)$, $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$, or $\hat{\boldsymbol{v}}^j$ to calculate, it is possible to adaptively select a wavelet family for each node to achieve the sparsest $\hat{\boldsymbol{v}}^j$ every iteration. To this authors knowledge, this has not been done with any SSWA method in the literature. The residual associated with calculating $\Delta\hat{\boldsymbol{v}}_\Delta^{j+1}$ with Equation (5.27) is:

$$\|\bar{\boldsymbol{d}}_\Delta\|_2 = \| - \bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j) - \bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1} \cdot \Delta\hat{\boldsymbol{v}}_\Delta^{j+1}\|_2. \tag{5.31}$$

A new approach that takes advantage of the above formulation is explored in this section. This approach, which utilizes a hybrid time-wavelet approach, is called Hybrid Reduced Column Steady-State Wavelet Analysis (HRCSSWA) and is shown in Algorithm 5.3. In Algorithm 5.3, $\bar{\boldsymbol{u}}$ is a vector with the same number of elements as $\bar{\boldsymbol{v}}^j$ with all elements set to 1, Support$(\boldsymbol{W}\bar{\boldsymbol{v}}^j)$ is the list of all non-zero elements left in $\boldsymbol{W}\bar{\boldsymbol{v}}^j$ after it has been thresholded using h, and LUSolve $\left(\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}, \hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)\right)$ is a function that performs the LU factorization described in Section 5.2 on $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}$ and calculates $\Delta\hat{\boldsymbol{v}}_g$ using L_1, U, and $\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j)$. SparseSearch $\left(S_v^{j+1}, S_v^j\right)$ represents the method for finding the sparsest wavelet domain representation of S_Δ^j.

The sparse search is performed by taking the waveforms for each node in $\bar{\boldsymbol{g}}_v^j$ and $\bar{\boldsymbol{v}}^j$ and dividing them into their individual nodal variable waveforms: $\bar{g}_{v,k}$ and $\bar{v}_k^j$. For each node, $\bar{g}_{v,k}$ and $\bar{v}_k^j$ are transformed into the wavelet domain using the wavelets from selected wavelet families (i.e. transformed into $\hat{g}_{v,k}$ and

[I] $\boldsymbol{W}_\Delta^{-1}$ can be applied right before Equation (5.27) is solved rather than including it in the formation of the Jacobian matrix.

$\hat{v}_k^j$). Next, $\hat{g}_{v,k}$ and $\hat{v}_k^j$ are thresholded using Equation (5.14) and the wavelet that provides the sparsest wavelet domain representations of $\hat{g}_{v,k}$ and $\hat{v}_k^j$ combined is selected for use with node k and the support of $\hat{g}_{v,k}$ and $\hat{v}_k^j$ are added to S_Δ^j for node k.

Once the support and list of wavelets for all nodes have been found, $\boldsymbol{W}^{-1}$ is then created by using the wavelet transform matrices for the selected wavelets. Since $\boldsymbol{W}^{-1}$ is block diagonal with each block being comprised of an inverse wavelet transform matrix, it is easy to substitute a separate wavelet transform matrix into each block. Lastly, $\boldsymbol{W}_\Delta^{-1}$ is created out of the columns in $\boldsymbol{W}^{-1}$ that are indexed by S_Δ^j.

Algorithm 5.3 HRCSSWA Algorithm

$j \leftarrow 0$

$\bar{\boldsymbol{v}}^0 \leftarrow$ Initial guess.

$\Delta\bar{\boldsymbol{v}}^1 \leftarrow -\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^0)^{-1}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^0)$

$\bar{\boldsymbol{v}}^1 \leftarrow \bar{\boldsymbol{v}}^0 + \Delta\bar{\boldsymbol{v}}^1$

repeat

$j \leftarrow j + 1$

$S_\Delta^j \leftarrow \text{SparseSearch}\left(S_v^{j+1}, S_v^j\right)$

Form $\boldsymbol{W}_\Delta^{-1}$

// Calculate $\Delta\hat{\boldsymbol{v}}_g$ with LU decomposition.

$\Delta\hat{\boldsymbol{v}}_g \leftarrow \text{LUSolve}\left(\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}, -\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j)\right)$

$\|\bar{\boldsymbol{d}}_\Delta\|_2 \leftarrow \| - \bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j) - \bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1} \cdot \Delta\hat{\boldsymbol{v}}_g\|_2$

if $\left(\|\bar{\boldsymbol{d}}_\Delta\|_2 > \beta\right)$ **or** $\left(\|\bar{\boldsymbol{d}}_\Delta\|_2 > \alpha \cdot \|\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j)\|_2\right)$ **then**

// The conditions in Equation (5.21) are not met,

// calculate $\Delta\bar{\boldsymbol{v}}^{j+1}$ in the time domain

// intsead.

$\Delta\bar{\boldsymbol{v}}^{j+1} \leftarrow -\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)^{-1}\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^j)$

else

$\Delta\bar{\boldsymbol{v}}^{j+1} \leftarrow \boldsymbol{W}_\Delta^{-1}\Delta\hat{\boldsymbol{v}}_g$

end if

// Update Guess

$\bar{\boldsymbol{v}}^{j+1} \leftarrow \bar{\boldsymbol{v}}^j + \Delta\bar{\boldsymbol{v}}^{j+1}$

$\bar{\boldsymbol{n}} \leftarrow r_{tol}\min\{|\bar{\boldsymbol{v}}^j|, |\bar{\boldsymbol{v}}^{j+1}|\} + a_{tol}\bar{\boldsymbol{u}}$

until $\Delta\bar{\boldsymbol{v}}_k^{j+1} < \bar{\boldsymbol{n}}_k\,, \forall k \in \{1, 2, \ldots, NM\}$ **and** $\|\bar{\boldsymbol{f}}(\bar{\boldsymbol{v}}^{j+1})\|_\infty < a_{tol}$

The formulation used for HRCSSWA removes the need for the pre-multiplication by $\boldsymbol{W}$ from the

circuit equations and reduces the amount of computational overhead required for the remaining wavelet transform since $\boldsymbol{W}_{\Delta}^{-1}$ will have fewer columns than $\boldsymbol{W}$. Additionally, the wavelet domain is only required for the calculation of $\Delta\hat{\boldsymbol{v}}^{j+1}$ and all other steps can be treated the same as for SSTD. There is however, additional computational overhead from performing the wavelet transforms on $\hat{g}_{v,k}$ and $\hat{v}_k^j$ for each node as well as the overhead of forming $\boldsymbol{W}_{\Delta}^{-1}$ each iteration.

5.3.1 Simulation Studies

The results of simulations run using the Cardoon circuit simulator for the circuits studied in Section 4.2 are presented in this section. The Haar, Daubechies, Coiflet, Symlet, biorthogonal and reverse biorthogonal families were used in this study. Conversion of $\bar{g}_v$ and $\bar{v}^j$ into $\hat{g}_v$ and $\hat{v}^j$ for the support search was achieved using the fast wavelet transform functions of the PyWavelets library which can be set to handle periodic waveforms. The simulations were performed with 512 coefficients, $a_{tol} = 10^{-7}$, $n_{tol} = 10^{-15}$, and $r_{tol} = 10^{-4}$. LU factorization of the Jacobian matrices were performed using the UMFPack routines in SuiteSparse [49] with row scaling disabled.

The HRCSSWA method is compared with SSWA and RCSSWA using the following metrics: simulation run times, calculation times, numeric factorization time, symbolic factorization time, number of iterations, average number of non-zero elements in the Jacobian matrices, percentage of non-zero entries in S_{Δ}^j each iteration. The average densities of the L_1, U, and L_2 matrices is also included. For the simulation time comparison, only the wavelets that led to the lowest run times with SSWA and RCSSWA are compared with HRCSSWA. Only the wavelets that provided the lowest average number of non-zero elements with SSWA and RCSSWA were used in the comparison of average Jacobian matrix density and number of non-zero elements. For the percentage of non-zero elements in S_{Δ}^j comparison, the wavelets that provided the lowest percentage of non-zero elements in S_{Δ}^j and average lowest number of non-zero elements in the Jacobian matrix with RCSSWA were compared with HRCSSWA. The wavelets that were utilized by each simulation are also presented along with the percentage of times each was utilized so it can be determined if any wavelet family tends to provide the sparsest representations for each of the circuits in the study.

For HRCSSWA, the calculation time is given by:

$$T_{hrc} = \sum_j T_{HS}^j + T_{\bar{v}\Delta}^j + T_{\Delta\bar{v}}^j, \tag{5.32}$$

where T_{HS}^j is the HRCSSWA overhead time; $T_{\Delta\bar{v}}^j$ is the time taken to calculate $\Delta\bar{\boldsymbol{v}}^{j+1}$ using Equation (2.18); and $T_{\bar{v}\Delta}^j$ is the time taken to calculate $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ using Equation (5.27) and transform it into the time domain. On iterations when the conditions in Equation (5.21) are met, the calculation time is determined by the time taken to find and test $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ and, therefore, $T_{\Delta\bar{v}}^j = 0$. The overhead time for the HRCSSWA algorithm is equal to the time taken to find S_{Δ}^j, form $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_{\Delta}^{-1}$, and test $\|\hat{\boldsymbol{d}}_{\Delta}\|_2$ on iterations when

the conditions in Equation (5.21) are met. Otherwise, the overhead time includes time spent calculating $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$ and is equal to $T_{HS}^{j} + T_{\bar{v}\Delta}^{j}$.

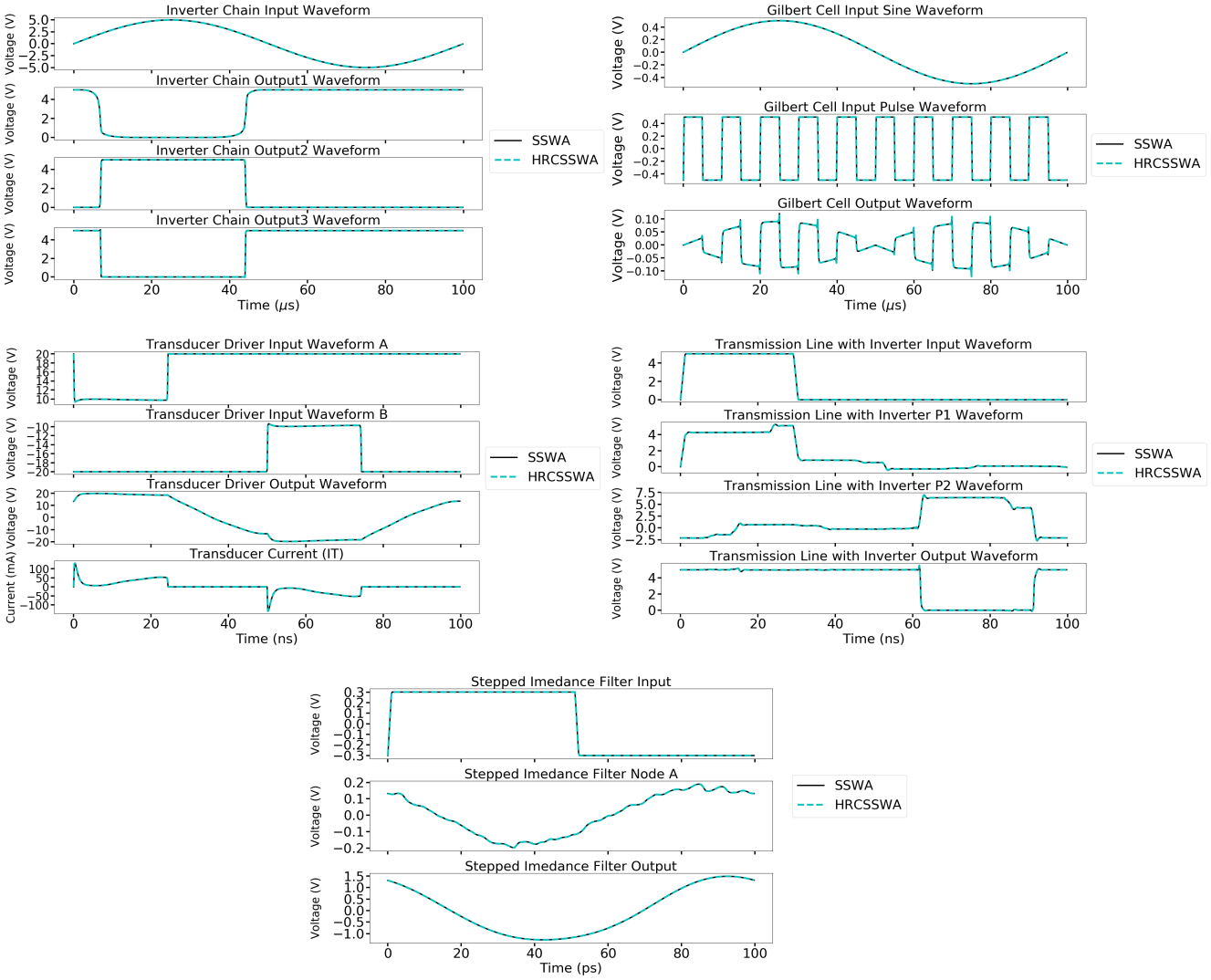

Figure 5.21: Input and output nodal variable waveforms for the five test circuits with HRCSSWA vs SSWA.

The input and output waveforms for the five test circuits with SSWA and HRCSSWA are shown in Figure 5.21. There is no noticeable difference between SSWA and HRCSSWA in the final calculated $\bar{\boldsymbol{v}}$ waveforms.

A list of all wavelets used by HRCSSWA, out of 105 wavelets within the selected families available with the PyWavelets library, for each simulation is shown in Table 5.6. The percentage usage is calculated as the total number of times a wavelet was utilized divided by the total number of times any wavelet was utilized. Most of the circuits utilized wavelets from multiple wavelet families with a varying number

of vanishing moments but, even so, a specific wavelet was favoured over the others in each simulation. For this reason, the two most commonly utilized wavelets for each circuit were used to create a new, shorter list for HRCSSWA and the simulations were run again. This approach has the advantage of only having to search a shorter list which saves time during the support search since less wavelets are checked during each iteration. The reduced column Jacobian matrix was not used for any of the iterations with HRCSSWA during the stepped impedance filter circuit simulation and, therefore, no wavelets are listed for this circuit. The wavelets used in the second set of HRCSSWA simulations were the Haar, Db6, Bior3.1, Bior3.3, and Bior5.5 wavelets. The simulations that utilized the shorter list of wavelets are recorded as 'limited wavelets' in the presented simulation results.

The simulation timing data for the five test circuits with HRCSSWA compared to SSWA and RCSSWA with the wavelets that provided the lowest simulation run times is shown in Table 5.7. Due to the significant reduction in support search time, HRCSSWA with the limited wavelets list ran faster than HRCSSWA with all wavelets in every test case except for the Gilbert cell circuit where the extra iteration caused a slowdown with the limited wavelets.

Table 5.6: Percentage of times each wavelet was selected during the simulations for the five test circuits. Blue font indicates two most commonly used wavelets for each circuit.

	Inverter Chain	Gilbert Cell	Ultrasound Transducer Driver	Transmission Line Circuit	Stepped Impedance Filter
Bior2.2	-	1.67	-	0.89	-
Bior3.1	67.97	1.67	62.50	64.00	-
Bior3.3	8.23	-	-	4.89	-
Bior3.5	4.76	-	-	-	-
Bior3.7	0.87	-	-	-	-
Bior4.4	0.43	-	-	-	-
Bior5.5	3.90	6.67	-	-	-
Db2	-	3.33	-	-	-
Db3	-	-	-	-	-
Db4	-	-	-	-	-
Db5	-	-	-	-	-
Db6	-	6.67	-	-	-
Haar	5.19	80.00	37.50	30.22	-
Rbio1.5	3.46	-	-	-	-
Rbio2.4	0.43	-	-	-	-
Rbio2.6	0.43	-	-	-	-
Rbio3.7	0.43	-	-	-	-
Sym4	0.43	-	-	-	-
Sym5	2.16	-	-	-	-
Sym6	0.43	-	-	-	-
Sym7	0.43	-	-	-	-
Sym8	-	-	-	-	-
Sym11	0.43	-	-	-	-

Table 5.7: Simulation timing data and number of iterations for for the five test circuits. Green font indicates where the HRCSSWA method with the limited list of wavelets is faster than the SSWA method. Blue font indicates where the RCSSWA method is faster than HRCSSWA with the limited list of wavelets.

	Inverter Chain	Gilbert Cell	Ultrasound Transducer	Transmission Line	Stepped Impedance Filter
SSWA	43.35	17.60	16.76	86.89	246.5
Sim Time	[1.204]	[2.514]	[1.862]	[2.633]	[41.09]
RCSSWA	38.09	20.39	15.07	84.75	262.1
Sim Time	[1.190]	[2.548]	[1.507]	[2.493]	[37.44]
HRCSSWA	49.71	22.94	15.67	159.4	427.2
Sim Time	[1.344]	[2.868]	[1.205]	[4.689]	[71.20]
HRCSSWA	38.33	23.44	13.60	146.9	353.5
Sim Time (limited wavelets)	[1.198]	[2.604]	[1.046]	[4.590]	[58.92]
SSWA	5.153	8.572	13.55	66.25	239.3
Calc Time	[0.143]	[1.225]	[1.506]	[2.008]	[39.89]
RCSSWA	6.367	10.79	11.24	61.21	248.4
Calc Time	[0.187]	[1.349]	[1.124]	[1.800]	[35.48]
HRCSSWA	16.66	15.49	12.11	146.7	421.8
Calc Time	[0.450]	[1.937]	[0.931]	[4.315]	[70.29]
HRCSSWA	10.12	15.50	10.06	134.4	348.2
Calc Time (limited wavelets)	[0.316]	[1.723]	[0.774]	[4.199]	[58.04]
SSWA Iterations	36	7	9	33	6
RCSSWA Iterations	32	8	10	34	5
HRCSSWA Iterations	37	8	13	34	6
HRCSSWA Iterations (limited wavelets)	32	9	13	32	6
RCSSWA	0.345	1.373	0.682	4.559	33.64
Overhead Time	[0.011]	[0.172]	[0.068]	[0.134]	[4.805]
HRCSSWA	11.04	7.708	4.073	72.13	131.0
Overhead Time	[0.299]	[0.964]	[0.313]	[2.122]	[21.84]
Jacobian Matrix	5.264	3.710	1.308	41.76	63.85
Formation Time	[0.142]	[0.464]	[0.101]	[1.228]	[10.64]
Support Search	5.531	2.235	2.031	13.26	7.839
Time	[0.150]	[0.279]	[0.156]	[0.390]	[1.306]
HRCSSWA	5.115	7.620	2.156	59.45	98.60
Overhead Time	[0.160]	[0.847]	[0.166]	[1.858]	[16.434]
(limited wavelets)					
Jacobian Matrix	4.599	4.383	1.326	46.78	58.83
Formation Time	[0.144]	[0.487]	[0.102]	[1.462]	[9.806]
(limited wavelets)					
Support Search Time	0.255	0.131	0.116	0.822	0.528
(limited wavelets)	[0.008]	[0.015]	[0.009]	[0.026]	[0.088]
Note: Calculated per-iteration values shown in square brackets and all times are in seconds.					

HRCSSWA with the limited list of wavelets exhibited comparable simulation run times to SSWA and RCSSWA for the inverter chain, Gilbert cell, and ultrasound transducer driver and exhibited speedups

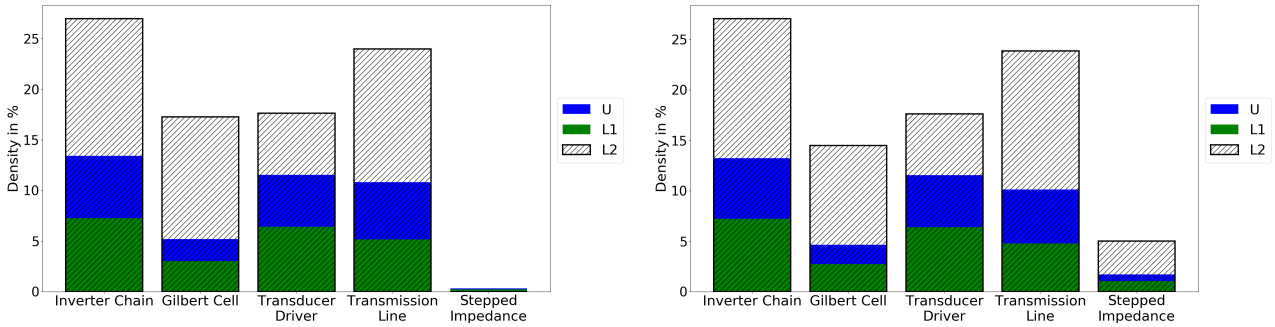

Figure 5.22: Average density of the L_1, U, and L_2 matrices for the simulations with all wavelet families (left) and the limited list of wavelets (right).

over SSWA with the inverter chain and ultrasound transducer driver circuits. Based on per-iteration simulation time, HRCSSWA with the limited list of wavelets would have run faster than SSTD if the number of iterations had remained unchanged. The simulation times for HRCSSWA with the limited wavelets are close to the lowest simulation times for SSWA and RCSSWA with all but the transmission line with inverter and stepped impedance filter circuits (see Tables 5.1, 5.2, 5.3, 5.4, and 5.5). Additionally, with the transmission line circuit, HRCSSWA with the limited wavelets list leads to lower simulation times than some of the wavelets with SSWA and RCSSWA. This indicates that although HRCSSWA with the limited list of wavelets did not provide the highest overall speeds in every test case, it could provide competitive speeds while also removing the problem of selection of most suitable wavelet that occurs with SSWA and RCSSWA. However, the reduced size Jacobian matrix was not used much with the stepped impedance filter circuit with HRCSSWA with the limited list of wavelets (and not at all for HRCSSWA with the full list of wavelets) which indicates there may be some problems that do not benefit from the use of HRCSSWA.

The support estimate time is significantly reduced by limiting the list of wavelets but, even with simulations that have a high number of iterations that did not utilize $\Delta\hat{\boldsymbol{v}}_{\Delta}^{j+1}$, the time taken to form $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_{\Delta}^{-1}$ is still significant. The overhead times for HRCSSWA are much higher than RCSSWA because the Jacobian matrix formation time is included with the overhead of support estimation[I]. The overhead times for HRCSSWA with the limited list of wavelets, without the Jacobian matrix formation times, are much higher than RCSSWA for three of the test circuits and close to the same for the others.

[I]With HRCSSWA, the Jacobian matrix is formed after the support estimation rather than before so the Jacobian matrix formation time is included with the overhead time calculation. Subtracting this time from the total overhead time allows for the comparison of the overhead times for support estimation, error checking, and recalculation of the Newton update vector (if the sparse estimate is not utilized) between RCSSWA and HRCSSWA.

Table 5.8: Matrix factorization timing data and number of iterations for for the five test circuits. Green font indicates where the HRCSSWA method with the limited list of wavelets is faster than the SSWA method. Blue font indicates where the RCSSWA method is faster than HRCSSWA with the limited list of wavelets.

		Inverter Chain	Gilbert Cell	Ultrasound Transducer	Transmission Line	Stepped Impedance Filter
SSWA	Symbolic	0.295	0.797	0.406	4.183	9.553
	Time	[0.008]	[0.114]	[0.045]	[0.127]	[1.592]
	Numeric	0.926	5.021	10.90	43.40	190.4
	Time	[0.026]	[0.717]	[1.211]	[1.315]	[31.74]
RCSSWA	Symbolic	0.295	0.844	0.373	3.328	10.96
	Time	[0.008]	[0.106]	[0.037]	[0.098]	[1.566]
	Numeric	1.207	6.838	8.986	43.58	185.2
	Time	[0.033]	[0.855]	[0.899]	[1.282]	[26.46]
HRCSSWA	Symbolic	0.372	0.795	0.307	2.746	6.997
	Time	[0.010]	[0.099]	[0.024]	[0.081]	[1.166]
	Numeric	3.381	6.454	6.559	69.53	297.9
	Time	[0.091]	[0.807]	[0.505]	[2.045]	[49.66]
HRCSSWA (Lim Wave)	Symbolic	0.356	0.811	0.305	2.749	5.726
	Time	[0.011]	[0.090]	[0.024]	[0.086]	[0.954]
	Numeric	2.921	7.432	6.476	65.74	242.6
	Time	[0.091]	[0.826]	[0.498]	[2.054]	[40.43]

Note: Calculated per-iteration values shown in square brackets and all times are in seconds.

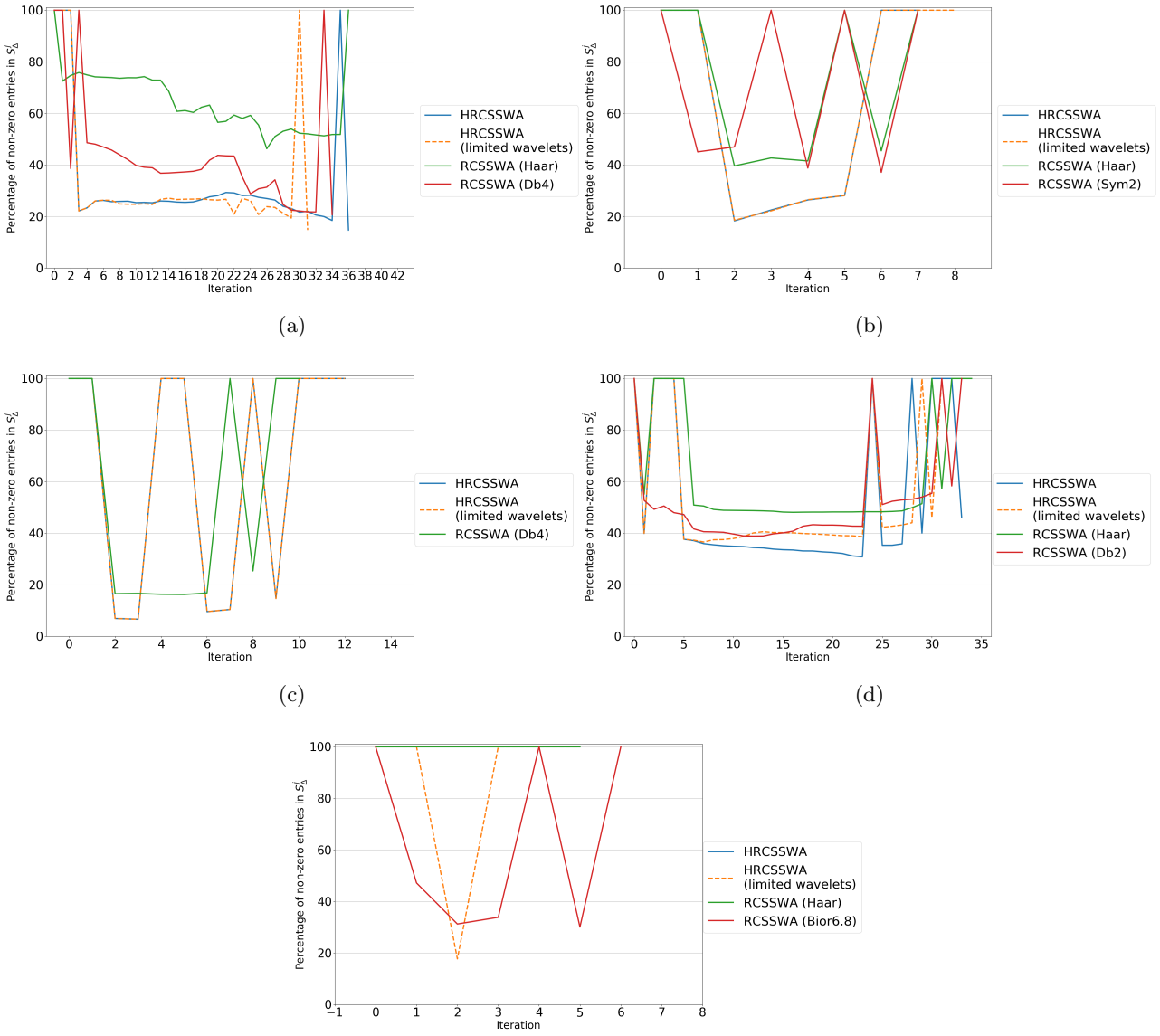

Figure 5.23: Percentage of non-zero elements in S_Δ^j for HRCSSWA with the full wavelet list and limited wavelet list (limited wavelets). Also shown for comparison are the RCSSWA Percentage of non-zero elements in S_Δ^j with the wavelet that provided the lowest average number of non-zero elements in the Jacobian matrices (green line) and the lowest number of non-zero elements in S_Δ^j (red line) for the (a) inverter chain, (b) Gilbert cell, (c) ultrasound transducer driver, (d) transmission line circuit, and (e) stepped impedance filter circuit.

In Figure 5.23, HRCSSWA is compared with the wavelets that provided the lowest number of non-zero elements in the Jacobian matrices and lowest average percentage of non-zero elements in S_Δ^j with RCSSWA. For most circuits HRCSSWA leads to a lower percentage of elements in S_Δ^j on average than

RCSSWA. For the stepped impedance filter circuit, there are few iterations that utilized the sparse estimate of $\hat{\boldsymbol{v}}^j$ with both RCSSWA (all but one wavelet) and HRCSSWA. For this reason, the simulation times using RCSSWA and HRCSSWA are not competitive with SSWA and SSTD for this circuit. This problem could potentially be addressed by improving the metric used to control $\|\hat{\boldsymbol{d}}_\Delta\|_2$ or the column selection method but, with the current formulation, there are certain circuits that do not benefit from the reduced column method. Utilizing the shorter list of wavelets leads to a small difference in S_Δ^j with HRCSSWA with most of the circuits. No change is seen with the ultrasound transducer driver since the two wavelets that were utilized with this circuit were both on the shortened wavelet list with the limited wavelets simulations.

The average density of the L_1, U, and L_2 matrices for each of the test circuits is shown in Figure 5.22. The average densities of the L_1, U, and L_2 matrices is not significantly different when using the shorter list of wavelets for most of the circuits.

5.3.2 Discussion

Allowing for a separate wavelet for each node leads to sparser representations of the nodal variable vectors and less computational overhead to form the Jacobian matrix than SSWA and RCSSWA. However, this comes at the cost of a higher overhead time to find the ideal combination of wavelets to use in the formation of $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}$ each iteration. This overhead is a significant disadvantage of utilizing HRCSSWA. The results with the test circuits show that the Bior3.1 and Haar wavelets were utilized to the highest extent with HRCSSWA with the Db6, Bior3.3, and Bior5.5 wavelets being the second most utilized wavelets. Utilizing just these wavelets, as was the case with the 'limited wavelets' simulations, leads to similar nodal variable vector and Jacobian matrix sparsities along with a significant reduction in overhead time associated with the support estimation compared to HRCSSWA when all wavelets are utilized. The limited list of wavelets was determined experimentally when running simulations with all 105 wavelets within the selected families available in the PyWavelets library. The list of 5 wavelets significantly reduced the overhead time associated with the support search of HRCSSWA. However, one could select a list that is two to three times larger and not significantly increase the support search times. Additionally, it is possible to select some of the wavelets based on the known input waveforms to a circuit. For example, the Haar wavelet will provide the sparsest representation of DC and square wave waveforms and should be included with a circuit that contains these inputs. Since many of the wavelets required to achieve the lowest density representations of S_Δ^j can be removed without having too large of an effect on the density of S_Δ^j, as is the case with the inverter chain, using a limited list of wavelets in this manner can increase the speed of HRCSSWA at the possible cost of a small increase in density of S_Δ^j.

The simulation times for HRCSSWA with the limited list of wavelets for some of these test cases are

higher than SSWA with the wavelet that led to the fastest simulation times. However, in the case of the Gilbert cell and transmission line with inverter circuits, HRCSSWA ran faster than SSWA with some of the other wavelets. This indicates that HRCSSWA could be utilized as an alternative to SSWA if the wavelet that will provide the best performance with SSWA is not known. This would result in a relatively small reduction in speed over the ideal case with SSWA but, considering the number of possible wavelets to choose from, the risk of selecting an inefficient wavelet justifies this cost in cases where selection of the most efficient wavelet for SSWA is difficult.

The densities of the L_1, U, and L_2 matrices are higher with HRCSSWA than RCSSWA. If the densities of these matrices is similar with larger circuit problems then the LU factorization algorithm may not be as efficient as RCSSWA with larger problems. The L_2 matrix, which exhibits densities over two times that of the L_1 matrix, may cause memory/computational efficiency issues with very large circuits. However, testing with large circuits would be necessary to confirm if this is the case.

The percentage of non-zero elements in S_{Δ}^{j} with HRCSSWA is lower than with RCSSWA. This indicates that assigning a wavelet to each nodal variable adaptively successfully leads to sparser nodal variable vectors and fewer columns in the reduced column Jacobian matrix than selecting one wavelet for all nodal variables to use for the entire simulation.

5.4 Chapter Summary

This chapter presented methods for reducing the columns in the Jacobian matrix at each iteration. The results show that the methods are successful in reducing the number of columns in the Jacobian matrix at each iteration. The HRCSSWA algorithm requires a higher calculation overhead per iteration to perform wavelet selection which can be mitigated by limiting the list of wavelets used in the search. RCSSWA and HRCSSWA with the limited list of wavelets led to a speedup over SSWA in some of the test cases. In all but one of the simulations where HRCSSWA with the limited list of wavelets ran slower than SSWA, some of the other wavelets with SSWA led to slower simulations than HRCSSWA with the limited list of wavelets. This indicates that HRCSSWA with a limited list of wavelets could be utilized instead of SSWA in cases where the most efficient wavelet for SSWA is not known. The trade off is a reduction in speed compared to SSWA when the optimum wavelet is selected but at a gain of not risking an even higher reduction in speed when an inefficient wavelet is selected. For the stepped impedance filter circuit with RCSSWA and HRCSSWA, the reduced column Jacobian was not utilized very often. This indicates that there are some problems that do not benefit from the column reduction methods in their current state. This problem can potentially be corrected by improving the criterion that controls $\|\hat{\boldsymbol{d}}_{\Delta}\|_2$ and/or the support search method but more work is required to determine if this is the case.

The next Chapter discusses this, and other future directions for this research and also concludes this

thesis with a brief summary of the results of Chapters 4 and 5.

Chapter 6

Conclusions and Future Work

Presented in this thesis is a study on the effect wavelet selection has on the densities of the Jacobian matrix and nodal variable vectors of SSWA. Three algorithms that attempt to increase the efficiency of SSWA with Newton method were also developed and presented. In terms of Jacobian matrix density, the Haar wavelet provided the lowest densities of the wavelets tested with all test circuits but did not always provide the lowest $\hat{\boldsymbol{v}}$ vector densities. When Jacobian matrix thresholding was applied, the Haar wavelet still provided the lowest density Jacobian matrices for the majority, but not all, of the test cases. Thus, there appears to be no one wavelet that will always provide the sparsest Jacobian matrices but, if the most ideal wavelet is unknown, it can be safe to use the Haar wavelet if Jacobian density is a concern.

The adaptive Jacobian matrix thresholding algorithm was successful at reducing the average Jacobian matrix densities while rarely introducing a high enough error into the Newton update vectors to cause a significant change in total number of iterations required to converge. Adaptive thresholding led to a speedup over SSWA without thresholding in all but one test case and sometimes even led to a speedup over SSTD. Thus, applying an adaptive threshold to the Jacobian matrix can be used to increase the efficiency of SSWA. There is still the requirement of selecting the threshold increase and decrease factors, the initial threshold, and the absolute and relative tolerances. However, these values are not hard to determine since the absolute and relative tolerances can be determined from the characteristics of the simulation, the initial threshold can be set to any very small, but safe, value to avoid over thresholding the early iterations of the simulation and the values for the increase and decrease factors used in this study appear to be sufficient for the general case. It is possible to achieve lower simulation times than the adaptive thresholding method by careful selection of a static threshold. However, selection of an optimal threshold is difficult and could require trial and error. One problem that the adaptive threshold addresses is removing the need for a designer to select an ideal threshold. Thus, a potential slowdown, compared to an ideally selected static threshold, is the trade off of using the adaptive Jacobian matrix thresholding

method. However, considering the difficulty in selecting an ideal threshold, the adaptive Jacobian matrix thresholding method appears to be worth using if an ideal threshold is not known in advance.

The wavelets that achieved the greatest reduction in density were the ones that had the highest number of vanishing moments while the wavelets with the lowest number of vanishing moments, which are the ones that provided the greatest simulation speeds, achieved the lowest Jacobian densities. Based on the data from this study, there is no one wavelet that provides the greatest Jacobian matrix sparsity and speeds. However, the Haar wavelet tends to provide the sparsest Jacobian matrices when using SSWA with no thresholding. Therefore, the Haar wavelet would make a safe choice for speed/sparsity if the choice of wavelet for optimal performance is unknown.

The RCSSWA and HRCSSWA algorithms were developed to attempt to take advantage of the sparsity of the wavelet domain representations of the nodal variable vectors that were observed in the SSWA study. Both algorithms work within each iteration of Newton method to reduce the number of columns in the Jacobian matrix. This is achieved by first estimating the support of the unknown update vector, selecting columns from the Jacobian matrix using the estimated support, and then solving the new system of equations represented by the reduced column Jacobian matrix to provide a sparse estimate of the update vector. The error in the sparse estimate is tested and, if it is low enough, the estimate is used and, if it is too high, the estimate is rejected.

On iterations where the reduced column estimate was utilized, there was a significant decrease in the number of columns, and number of non-zero elements, in the Jacobian matrix with both RCSSWA and HRCSSWA. This resulted in a deviation between the update vectors calculated by SSWA, RCSSWA, and HRCSSWA that led to a change in the total number of iterations required to converge. The slowdown in simulation times was caused by an increase in the number of iterations and/or the extra overhead of implementing the column reduction methods. The overhead of HRCSSWA can be significantly reduced by reducing the number of wavelets that are used during support estimation. This also results in simulation times that are sometimes lower than the lowest simulation times that were achieved with SSWA. In cases where there was a slowdown, compared to the wavelet that provided the fastest simulation time, HRCSSWA still ran faster than some of the other wavelet choices. This indicates that HRCSSWA with a limited list of wavelets can be utilized in cases where the most efficient wavelet choice for SSWA is not known. The L_2 matrix will tend to exhibit densities that are much greater than the L_1 matrix. If the densities of the L_1, U, and L_2 matrices remain the same with larger circuit problems, this method may run into memory/computational efficiency issues when factorizing $\hat{\boldsymbol{J}}_{\Delta}$. However, further testing with larger circuit problems will be necessary to confirm if this is the case.

Adaptive Jacobian matrix thresholding provides the best increase in speed with the SSWA formulation

used in this thesis. However, with this formulation, SSTD still tends to provide the fastest simulation times. The waveforms with these circuits were not very sparse. RCSSWA and HRCSSWA have the potential to provide faster calculation speeds as the sparsity of the nodal variable waveforms increases and more work is required to determine if RCSSWA and HRCSSWA perform better with large, sparse, problems.

The speedups that occur with SSWA with adaptive thresholding and RCSSWA do not consistently happen with the same wavelets for every circuit, making wavelet selection difficult. HRCSSWA addresses this issue by automatically selecting a wavelet for each nodal variable to provide the sparsest representations of the nodal variable waveforms but at the price of a significant increase in overhead time. This overhead time can be reduced by limiting the list of wavelets used when estimating S_{Δ}^{j} but the overhead time of forming $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^{j})\boldsymbol{W}_{\Delta}^{-1}$, along with the increase in time required to factorize $\hat{\boldsymbol{J}}_{\Delta}$ compared to $\hat{\boldsymbol{J}}(\hat{\boldsymbol{v}}^{j})$ remain significant factors. The HRCSSWA simulations presented in this thesis exhibited slowdowns compared to SSTD. However, for the ultrasound transducer driver, the per-iteration simulation and numeric factorization times are lower for HRCSSWA than SSTD (see Tables 4.6 and 5.8). This could indicate that improvements to HRCSSWA could potentially lead to simulation times that are competitive with SSTD. However, in its present form, the HRCSSWA algorithm does not lead to speedups over SSTD with the examples in this thesis.

Jacobian matrix formation with HRCSSWA will be faster than RCSSWA since $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^{j})\boldsymbol{W}_{\Delta}^{-1}$ can be formed with far fewer operations than $\boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^{j})\boldsymbol{W}^{-1}$. Since Jacobian matrix formation is faster with HRCSSWA and the difficulty in selecting the proper wavelet for a given circuit is addressed by HRCSSWA, the focus of future research will be with HRCSSWA. There are three main avenues of future development for the HRCSSWA algorithm:

1. Increase the speed/efficiency of solving Equation (5.27).
2. Improvement of the estimate of S_{Δ}^{j}.
3. Increase the speed/efficiency of forming $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^{j})\boldsymbol{W}_{\Delta}^{-1}$.

One of the drawbacks of the LU factorization is the formation of matrix $\boldsymbol{L}_2$ of the partitioned problem. There is no guarantee that $\boldsymbol{L}_2$ will be sparse and, as such, it can potentially require a large amount of memory for storage. However, the $\boldsymbol{L}_2$ matrix is not required by the reduced column methods and, if an algorithm can be written that finds $\boldsymbol{L}_1$ and $\boldsymbol{U}$ without explicitly forming $\boldsymbol{L}_2$ can be developed, a significant decrease in memory resources could potentially be achieved. However, LU factorization requires extra memory for storage of the $\boldsymbol{L}$ and $\boldsymbol{U}$ matrices and may provide poor performance with very large problems. Therefore, an iterative method, such as GMRES [65] may be better suited to solving Equation (5.27) with

very large problems.

For the Gilbert cell and stepped impedance filter circuits, there were many iterations where the estimate of S_{Δ}^{j} led to a large enough error for the conditions in Equation (5.21) to not be met. This could be due to the conditions in Equation (5.21) being too aggressive. A new set of conditions may improve this method by allowing more iterations where $\Delta\hat{\boldsymbol{v}}_g$ is used as the Newton update vector. Additionally, a new thresholding method could be developed to better select elements to add to the support estimate.

The threshold used in this thesis is selected based on the waveforms being thresholded. The parameter r_{tol} is manually entered and was one of the parameters used to determine convergence of the simulation. However, a pre-selected threshold is not the only possible method. For example, threshold selection methods based on those used in data compression applications, such as those used in [66], could provide more optimal performance. However, a pre-selected threshold is the most computationally simple and care has to be taken to avoid adding too much computational overhead to the estimation of S_{Δ}^{j}.

The majority of the computational time of estimating S_{Δ}^{j} with HRCSSWA is due to the wavelet transforms required to find the most convenient wavelet. For each node, the waveforms are transformed into the wavelet domain, thresholded, and the number of non-zero elements in the thresholded vector are counted. In total, there were 105 wavelets used in the HRCSSWA study. This represents a very high amount of extra overhead that can be significantly reduced by utilizing a shorter list of wavelets. Based on this study, selection of a list of wavelets that will provide the greatest sparsity for all cases is not possible. However, one could select a single wavelet from each family (there were 6 used in the limited wavelets list of this study) which would significantly reduce the computational overhead associated with estimation of S_{Δ}^{j}. Based on this study, the Haar and Bior3.1 wavelets tended to be utilized the most. Additionally, the wavelets that provided the sparsest nodal variable vectors tended to be the wavelets with the lowest number of vanishing moments. Thus, the Haar, Db2, Coif2, Sym2, and Bior3.3 wavelets could be utilized in the general case where it is uncertain which wavelets to use with HRCSSWA.

Formation of $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^{j})\boldsymbol{W}_{\Delta}^{-1}$ is another significant source of computational overhead for HRCSSWA. The transform matrix, $\boldsymbol{W}_{\Delta}^{-1}$, is created by using the fast wavelet transform to form only the columns in $\boldsymbol{W}_{\Delta}^{-1}$ that correspond to the entries in S_{Δ}^{j}. This requires many operations to form $\boldsymbol{W}_{\Delta}^{-1}$ every iteration. If the limited list of wavelets is decided before the simulation is run, it is possible to form these matrices and keep them stored in memory for use during the simulation. This is is easily implemented and results in the matrices only needing to be formed once at the cost of a little extra use of memory[I]. However, this method would scale poorly to problems that have a high number of samples for the nodal variables. Another way to increase efficiency of formation of $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^{j})\boldsymbol{W}_{\Delta}^{-1}$ is to apply the fast wavelet transforms

[I] $5M^2$ elements have to be stored in memory if the wavelets list has 5 entries.

directly to $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)$. This would avoid the extra memory use from storing the wavelet transform matrices and should provide a speedup over SSWA since $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}_\Delta^{-1}$ can be formed with much fewer operations than $\boldsymbol{W}\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)\boldsymbol{W}^{-1}$.

One potential application that was not explored in this thesis is the case where multiple simulations with the same circuit are performed. For example, if a Monte Carlo analysis is performed, many simulations with the same circuit with small changes to the values of circuit elements (resistances, capacitances, inductances, etc.) are run to predict the effect of variations in the elements due to causes such as environmental effects or manufacturing tolerances. HRCSSWA could be applied to this type of problem by running the first simulation with the designed parameters. During this simulation the ideal wavelets for each node could be determined. This list could then be utilized for subsequent simulations to avoid the requirement of searching for the ideal wavelet at later iterations. The iterations from the first simulation that required refinement could also be flagged and, for subsequent simulations, the Newton updates for the flagged iterations could be calculated without trying reduce the columns. Since refinement will result in utilizing $\bar{\boldsymbol{J}}(\bar{\boldsymbol{v}}^j)$ to calculate $\Delta\bar{\boldsymbol{v}}$ directly, the overhead time required to calculate $\Delta\hat{\boldsymbol{v}}_g$ for these iterations would be removed and result in a reduction in calculation overhead.

Appendix A

Derivative Estimation

Due to the capacitances and charges requiring the time derivatives of the voltages at each node to calculate, Newton method cannot be directly applied to Equation (2.8) or Equation (2.9). Therefore, a method for calculating or approximating the time derivatives during each iteration is necessary so the Jacobian matrix can be formed and next the Newton update can be calculated. Some common derivative estimate methods are the Forward Differences Method, Backward Differences Method, Central Differences Method, and Fourier derivative (derivatives calculated in the frequency domain) method. This appendix briefly introduces these derivative methods.

A.1 Forward Differences Method

This method begins with the Taylor series expansion of the function $x(t)$ about an arbitrary point t_l:

$$x(t) = x(t_l) + \frac{x'(t_l)(t-t_l)}{1!} + \frac{x''(t_l)(t-t_l)^2}{2!} + \frac{x'''(t_l)(t-t_l)^3}{3!} + \cdots, \tag{A.1}$$

where N is the number of terms in the series, a prime denotes differentiation with respect to t, and all derivatives are evaluated at t_l[42]. Setting $\Delta t = t - t_l$ in Equation (A.1) yields

$$x(t_l + \Delta t) = x(t_l) + \frac{x'(t_l)(\Delta t)}{1!} + \frac{x''(t_l)(\Delta t)^2}{2!} + \frac{x'''(t_l)(\Delta t)^3}{3!} + \cdots. \tag{A.2}$$

If $x(t)$ is a vector representing a periodic waveform that is discretized with M equally spaced time samples then Equation (A.2) becomes:

$$x_{l+1} = x_l + \frac{x'_l(\Delta t)}{1!} + \frac{x''_l(\Delta t)^2}{2!} + \frac{x'''_l(\Delta t)^3}{3!} + \cdots, \tag{A.3}$$

where x_l is the value of $x(t)$ at time sample l, x_{l+1} is the value of $x(t)$ at time sample $l+1$, and x'_l is the value of the estimate of the derivative of $x(t)$ at time sample l.

To approximate the first order derivative, all terms after the first order derivative in Equation (A.3) are truncated:

$$x_{l+1} \approx x_l + \frac{x'_l(\Delta t)}{1!}. \tag{A.4}$$

The equation is then re-ordered to solve for x'_l:

$$x'_l \approx \frac{x_{l+1} - x_l}{\Delta t}. \tag{A.5}$$

Equation (A.5), known as the 2 point forward differences method, can be expressed using matrix vector products as:

$$\begin{bmatrix} x'_0 \\ x'_1 \\ \vdots \\ x'_M \end{bmatrix} \approx \frac{1}{\Delta t} \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix}, \tag{A.6}$$

$$x'(t) \approx D_{F2} x(t). \tag{A.7}$$

The final row requires a sample that is outside the time period (sample $M+1$) to calculate the derivative x'_M. Since $x(t)$ is periodic, the sample that falls in the next time period can be found by wrapping around to the beginning of the row (i.e. sample $M+1$ is the same as sample 1).

The error of the 2 point forward differences method is the sum of the remaining terms from Equation (A.4) that were neglected. Equation (A.5) with all terms is:

$$x'_l = \frac{x_{l+1} - x_l}{\Delta t} - \frac{x''_l(\Delta t)}{2!} - \frac{x'''_l(\Delta t)^2}{3!} - \frac{x''''_l(\Delta t)^3}{4!} - \cdots, \tag{A.8}$$

and the remaining terms are:

$$R_{F2} = -\frac{x''_l(\Delta t)}{2!} - \frac{x'''_l(\Delta t)^2}{3!} - \frac{x''''_l(\Delta t)^3}{4!} - \cdots. \tag{A.9}$$

The lower order derivative terms usually contribute more to the error than the higher order derivative terms and R_{F2} can be approximated using just the first of the remaining terms [42]:

$$R_{F2} \approx -\frac{x''_l(\Delta t)}{2!}. \tag{A.10}$$

It is possible to increase the accuracy of the approximate derivative by using more datapoints in the Taylor series expansion of $x(t)$. For example, the 2 point method utilizes two points: x_{l+2} and x_{l+1}. The 2 point method begins with the first 3 terms of the Taylor series expansion in Equation (A.3) for x_{l+2} and x_{l+1}:

$$x_{l+2} \approx x_l + \frac{x'_l(2\Delta t)}{1!} + \frac{x''_l(2\Delta t)^2}{2!}, \tag{A.11}$$

$$x_{l+1} \approx x_l + \frac{x'_l(\Delta t)}{1!} + \frac{x''_l(\Delta t)^2}{2!}. \tag{A.12}$$

Since the time samples are equally spaced, the time step is twice as large for the expansion of x_{l+2} and Δt must be multiplied by 2 in Equation (A.11).

Multiplying Equation (A.12) by 4 and subtracting it from Equation (A.11) yields:

$$x_{l+2} - 4x_{l+1} \approx -3x_l - x'_l(2\Delta t). \tag{A.13}$$

The estimate of the 2 point derivative is then found by solving for x'_l Equation (A.13):

$$x'_l \approx \frac{-x_{l+2} + 4x_{l+1} - 3x_l}{2\Delta t}. \tag{A.14}$$

Equation (A.14) can be expressed using matrix vector products:

$$\begin{bmatrix} x'_0 \\ x'_1 \\ \vdots \\ x'_M \end{bmatrix} \approx \frac{1}{2\Delta t} \begin{bmatrix} -3 & 4 & -1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & -3 & 4 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -3 & 4 & -1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 4 & -1 & 0 & 0 & 0 & 0 & \cdots & -3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix}, \tag{A.15}$$

$$x'(t) \approx D_{F4}x(t). \tag{A.16}$$

The error of the 4 point forward differences method is the combination of the terms of Equations (A.11) and (A.12) that were not included in the estimate. Equations (A.11) and (A.12) with all terms are:

$$x_{l+2} = x_l + \frac{x'_l(2\Delta t)}{1!} + \frac{x''_l(2\Delta t)^2}{2!} + \frac{x'''_l(2\Delta t)^3}{3!} + \frac{x''''_l(2\Delta t)^4}{4!} + \cdots, \tag{A.17}$$

$$x_{l+1} = x_l + \frac{x'_l(\Delta t)}{1!} + \frac{x''_l(\Delta t)^2}{2!} + \frac{x'''_l(\Delta t)^3}{3!} + \frac{x''''_l(\Delta t)^4}{4!} + \cdots. \tag{A.18}$$

Multiplying Equation (A.18) by 4 and subtracting it from Equation (A.17) yields[I]:

$$x_{l+2} - 4x_{l+1} = -3x_l - x'_l(2\Delta t) + 4\frac{x'''_l(\Delta t)^3}{3!} + 12\frac{x''''_l(\Delta t)^4}{4!} + \cdots, \tag{A.19}$$

$$x'_l = \frac{-x_{l+2} + 4x_{l+1} - 3x_l}{2\Delta t} + 2\frac{x'''_l(\Delta t)^2}{3!} + 6\frac{x''''_l(\Delta t)^3}{4!} + \cdots. \tag{A.20}$$

The error is given by the remaining terms that were not used in Equation (A.14):

$$R_{F4} = 2\frac{x'''_l(\Delta t)^2}{3!} + 6\frac{x''''_l(\Delta t)^3}{4!} + \cdots. \tag{A.21}$$

Which, due to having higher order derivatives than Equation (A.10), can be expected to be lower than the error of the 2 point method.

[I] These are the same operations that were performed on Equations (A.11) and (A.12) to form Equation (A.13).

A.2 Backward Differences

This method begins with a similar formulation as the Forward differences method but, instead of looking forward in time the method looks backward in time and $-\Delta t = t - t_l$ [42]. Substituting $-\Delta t = t - t_l$ into Equation (A.1) yields:

$$x(t_l - \Delta t) = x(t_l) + \frac{x'(t_l)(-\Delta t)}{1!} + \frac{x''(t_l)(-\Delta t)^2}{2!} + \frac{x'''(t_l)(-\Delta t)^3}{3!} + \cdots . \tag{A.22}$$

If $x(t)$ is a vector representing a periodic waveform that is discretized with M equally spaced time samples then Equation (A.22) becomes:

$$x_{l-1} = x_l + \frac{x'_l(-\Delta t)}{1!} + \frac{x''_l(-\Delta t)^2}{2!} + \frac{x'''_l(-\Delta t)^3}{3!} + \cdots , \tag{A.23}$$

where x_l is the value of $x(t)$ at time sample l, x_{l-1} is the value of $x(t)$ at time sample $l-1$, and x'_l is the value of the estimate of the derivative of $x(t)$ at time sample l.

To approximate the first order derivative, all terms after the first order derivative in Equation (A.23) are truncated:

$$x_{l-1} \approx x_l + \frac{x'_l(-\Delta t)}{1!}, \tag{A.24}$$

and the equation is re-ordered to solve for x'_l:

$$x'_l \approx \frac{x_l - x_{l-1}}{\Delta t}. \tag{A.25}$$

Equation (A.25) can be expressed using matrix vector products:

$$\begin{bmatrix} x'_0 \\ x'_1 \\ \vdots \\ x'_M \end{bmatrix} \approx \frac{1}{\Delta t} \begin{bmatrix} 1 & 0 & 0 & & \cdots & -1 \\ -1 & 1 & 0 & & \cdots & 0 \\ 0 & -1 & 1 & 0 & \cdots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & -1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix}, \tag{A.26}$$

$$x'(t) \approx D_{B2}x(t). \tag{A.27}$$

The first row requires a sample that is outside the time period (sample -1) to calculate the derivative x'_1. Since $x(t)$ is periodic, the sample that falls in the previous time period can be found by wrapping around to the end of the row (i.e. sample -1 is the same as sample M).

The error of the 2 point backward differences method is the sum of the remaining terms from Equation (A.24) that were neglected. Equation (A.25) with all terms is:

$$x'_l = \frac{x_l - x_{l-1}}{\Delta t} + \frac{x''_l(\Delta t)}{2!} - \frac{x'''_l(\Delta t)^2}{3!} + \frac{x''''_l(\Delta t)^3}{4!} - \cdots , \tag{A.28}$$

and the remaining terms are:

$$R_{B2} = \frac{x_l''(\Delta t)}{2!} - \frac{x_l'''(\Delta t)^2}{3!} + \frac{x_l''''(\Delta t)^3}{4!} - \cdots . \tag{A.29}$$

The lower order derivative terms usually contribute more to the error than the higher order derivative terms and R_{B2} can be approximated using just the first of the remaining terms [42]:

$$R_{B2} \approx \frac{x_l''(\Delta t)}{2!}. \tag{A.30}$$

It is possible to increase the accuracy of the approximate derivative by using more datapoints in the Taylor series expansion of $x(t)$. For example, the 2 point method utilizes two points: x_{l-1} and x_{l-2}. The 2 point method begins with the first 3 terms of the Taylor series expansion in Equation(A.23) for x_{l-1} and x_{l-2}:

$$x_{l-2} \approx x_l + \frac{x_l'(-2\Delta t)}{1!} + \frac{x_l''(-2\Delta t)^2}{2!}, \tag{A.31}$$

$$x_{l-1} \approx x_l + \frac{x_l'(-\Delta t)}{1!} + \frac{x_l''(-\Delta t)^2}{2!}. \tag{A.32}$$

Since the time samples are equally spaced, the time step is twice as large for the expansion of x_{l-2} and Δt must be multiplied by 2 in Equation (A.31).

Multiplying Equation (A.32) by 4 and subtracting it from Equation (A.31) yields:

$$x_{l-2} - 4x_{l-1} \approx -3x_l + x_l'(2\Delta t). \tag{A.33}$$

The estimate of the 2 point derivative is then found by solving for x_l' Equation (A.33):

$$x_l' \approx \frac{x_{l-2} - 4x_{l-1} + 3x_l}{2\Delta t}. \tag{A.34}$$

Equation (A.34) can be expressed using matrix vector products:

$$\begin{bmatrix} x_0' \\ x_1' \\ \vdots \\ x_M' \end{bmatrix} \approx \frac{1}{2\Delta t} \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & -4 \\ -4 & 3 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & -4 & 3 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -4 & 3 & 0 & \cdots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 & -4 & 3 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix}, \tag{A.35}$$

$$x'(t) \approx D_{B4}x(t). \tag{A.36}$$

The error of the 4 point backward differences method is the combination of the terms of Equations (A.31) and (A.32) that were not included in the estimate. Equations (A.31) and (A.32) with all terms are:

$$x_{l-2} = x_l + \frac{x_l'(-2\Delta t)}{1!} + \frac{x_l''(-2\Delta t)^2}{2!} + \frac{x_l'''(-2\Delta t)^3}{3!} + \frac{x_l''''(-2\Delta t)^4}{4!} + \cdots , \tag{A.37}$$

$$x_{l-1} = x_l + \frac{x'_l(-\Delta t)}{1!} + \frac{x''_l(-\Delta t)^2}{2!} + \frac{x'''_l(-\Delta t)^3}{3!} + \frac{x''''_l(-\Delta t)^4}{4!} \cdots . \tag{A.38}$$

Multiplying Equation (A.38) by 4 and subtracting it from Equation (A.37) yields[I]:

$$x_{l-2} - 4x_{l-1} = -3x_l - 2x'_l(-\Delta t) + 4\frac{x'''_l(-\Delta t)^3}{3!} + 12\frac{x''''_l(-\Delta t)^4}{4!} + \cdots, \tag{A.39}$$

$$x'_l = \frac{x_{l-2} - 4x_{l-1} + 3x_l}{2\Delta t} + 2\frac{x'''_l(\Delta t)^2}{3!} - 6\frac{x''''_l(\Delta t)^3}{4!} + \cdots. \tag{A.40}$$

The error is given by the remaining terms that were not used in Equation (A.34):

$$R_{B4} = 2\frac{x'''_l(\Delta t)^2}{3!} - 6\frac{x''''_l(\Delta t)^3}{4!} + \cdots. \tag{A.41}$$

Which, due to having higher order derivatives than Equation (A.30), can be expected to be lower than the error of the 2 point method.

A.3 Central Differences

This method is a combination of the Backward and Forward differences method. If Equation (A.23) is subtracted from Equation (A.3), the result is:

$$x_{l+1} - x_{l-1} = x_l - x_l + \frac{x'_l(\Delta t)}{1!} - \frac{x'_l(-\Delta t)}{1!} + \frac{x''_l(\Delta t)^2}{2!} - \frac{x''_l(-\Delta t)^2}{2!} + \cdots, \tag{A.42}$$

$$x_{l+1} - x_{l-1} = \frac{2x'_l(\Delta t)}{1!} + \frac{2x'''_l(\Delta t)^3}{3!} + \frac{2x''''_l(\Delta t)^5}{5!} + \cdots. \tag{A.43}$$

To approximate the first order derivative, all terms after the first order derivative in Equation (A.43) are truncated:

$$x_{l+1} - x_{l-1} \approx \frac{2x'_l(\Delta t)}{1!}, \tag{A.44}$$

and the equation is re-ordered to solve for x'_l:

$$x'_l \approx \frac{x_{l+1} - x_{l-1}}{2\Delta t}. \tag{A.45}$$

Equation (A.45) can be expressed using matrix vector products:

$$\begin{bmatrix} x'_0 \\ x'_1 \\ \vdots \\ x'_M \end{bmatrix} \approx \frac{1}{2\Delta t} \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \cdots & -1 \\ -1 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & 1 & 0 & \cdots & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & 0 & 0 & \cdots & -1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix}, \tag{A.46}$$

$$x'(t) \approx D_{C2}x(t). \tag{A.47}$$

[I]These are the same operations that were performed on Equations (A.31) and (A.32) to form Equation (A.33).

This is known as the 2 point central differences method. Since $x(t)$ is periodic, any required sample that is outside the time period can be obtained by wrapping around to the opposite side of the time period. For example, x_{-1} is the same as x_M and x_{M+1} is the same as x_0.

The error of the 2 point central differences method is the sum of the remaining terms from Equation (A.44) that were neglected. Equation (A.45) with all terms is:

$$x'_l = \frac{x_{l+1} - x_{l-1}}{2\Delta t} + \frac{x'''_l(\Delta t)^2}{3!} + \frac{x'''''_l(\Delta t)^4}{5!} + \cdots. \tag{A.48}$$

and the remaining terms are:

$$R_{C2} = \frac{x'''_l(\Delta t)^2}{3!} + \frac{x'''''_l(\Delta t)^4}{5!} + \cdots. \tag{A.49}$$

The lower order derivative terms usually contribute more to the error than the higher order derivative terms and R_{C2} can be approximated using just the first of the remaining terms [42]:

$$R_{C2} \approx \frac{x'''_l(\Delta t)^2}{3!}. \tag{A.50}$$

Which can be expected to be lower than the 2 point forward and backward differences approximations which contain lower order derivative terms.

It is also possible to increase the accuracy of the approximate derivative by using more datapoints in the Taylor series expansion of $x(t)$. For example, the 4 point central differences method can be derived by subtracting the Taylor series expansion of x_{l-2} from the Taylor series expansion of x_{l+2}[I]:

$$x_{l+2} - x_{l-2} = \frac{2x'_l(2\Delta t)}{1!} + \frac{2x'''_l(2\Delta t)^3}{3!} + \frac{2x'''''_l(2\Delta t)^5}{5!} + \cdots. \tag{A.51}$$

Multiplying Equation (A.43) by 8, subtracting it from Equation (A.51), and truncating all but the first and third derivative terms from the result yields:

$$x_{l+2} - x_{l-2} - 8x_{l+1} + 8x_{l-1} \approx \frac{2x'_l(2\Delta t)}{1!} + \frac{2x'''_l(2\Delta t)^3}{3!} - 8\frac{2x'_l(\Delta t)}{1!} - 8\frac{2x'''_l(\Delta t)^3}{3!}, \tag{A.52}$$

$$x_{l+2} - 8x_{l+1} + 8x_{l-1} - x_{l-2} \approx -12x'_l(\Delta t). \tag{A.53}$$

The estimate of the 4 point derivative is then found by solving for x'_l Equation (A.53):

$$x'_l \approx \frac{-x_{l+2} + 8x_{l+1} - 8x_{l-1} + x_{l-2}}{12\Delta t}. \tag{A.54}$$

[I]The first three terms of the Taylor series expansion of x_{l+2} and x_{l-2} are given in Equations (A.11) and (A.31) respectively.

Equation (A.54) can be expressed using matrix vector products:

$$\begin{bmatrix} x'_0 \\ x'_1 \\ \vdots \\ x'_M \end{bmatrix} \approx \frac{1}{12\Delta t} \begin{bmatrix} 0 & 8 & -1 & 0 & 0 & 0 & \cdots & 0 & 1 & -8 \\ -8 & 0 & 8 & -1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & -8 & 0 & 8 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & -8 & 0 & 8 & -1 & \ddots & \vdots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ 8 & -1 & 0 & 0 & 0 & 0 & \cdots & 1 & -8 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_M \end{bmatrix}, \tag{A.55}$$

$$x'(t) \approx D_{C4}x(t). \tag{A.56}$$

The error of the 4 point central differences method is the combination of the terms of Equations (A.51) and (A.43) that were not included in the estimate. Equation (A.54) with all terms is:

$$x'_l = \frac{-x_{l+2} + 8x_{l+1} - 8x_{l-1} + x_{l-2}}{12\Delta t} + 4\frac{x''''_l(\Delta t)^4}{5!} + 20\frac{x''''''_l(\Delta t)^6}{7!} + \cdots . \tag{A.57}$$

The error is given by the remaining terms that were not used in Equation (A.34):

$$R_{C4} = 4\frac{x''''_l(\Delta t)^4}{5!} + 20\frac{x''''''_l(\Delta t)^6}{7!} + \cdots \tag{A.58}$$

Which, due to having higher order derivatives than Equation (A.50), can be expected to be lower than the error of the 2 point method. This error also contains much higher order derivatives than the 4 point forward and backward differences methods and, therefore, can be expected to be lower than R_{F4} and R_{B4}.

A.4 Fourier (Frequency Domain) Differentiation Method

With the Fourier differentiation method, also known as a spectral differentiation [67] or pseudospectral method [68], the derivatives are performed in the frequency domain. These methods perform the derivative on a set of discrete data in three steps [67, 68]:

1. The discrete data $\bar{x}$ is transformed into the frequency domain with the Fourier Transform.

2. The derivative is performed by using the inverse discrete Fourier transform to define a continuous interpolation, $x_{df}(t)$, between the samples of the frequency domain representation of the discrete data [67]. The derivative of this interpolation is used to determine the derivatives of the frequency domain coefficients of the discrete data ($\tilde{u}$).

3. The derivative of $\tilde{u}$ is transformed into the time domain with the Fourier Transform to obtain the estimate of the derivatives of $\bar{x}$.

This method can be utilized for both periodic and non-periodic waveforms [67, 68]. However, since this thesis is concerned with periodic steady-state analysis, this Appendix will focus on periodic waveforms. For a description of how to use spectral differentiation for non-periodic waveforms, the reader is encouraged to review [67] and [68].

If a periodic waveform, x(t), with period T is discretized into M uniformly spaced samples over period T, the Fourier transform can be expressed as [67, 69]:

$$\tilde{u}_p = \frac{1}{M}\sum_{l=0}^{M-1} \bar{x}_l e^{-j\frac{2\pi}{M}lp}, \tag{A.59}$$

$$\bar{x}_l = \sum_{p=0}^{M-1} \tilde{u}_p e^{j\frac{2\pi}{M}lp}, \tag{A.60}$$

where $\bar{x}_l$ represents the l^{th} time sample, $\tilde{u}_p$ represents the p^{th} frequency coefficient, and $\bar{x}_l = x(\frac{lT}{M})$.. It is possible to construct matrices that perform the operations in Equations (A.59) and (A.60) and the transforms can be expressed using matrix/vector products:

$$\tilde{u} = \mathcal{F}\bar{x}, \tag{A.61}$$

$$\bar{x} = \mathcal{F}^{-1}\tilde{u}, \tag{A.62}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are matrices that perform the forward and inverse Fourier transforms, respectively, and $\tilde{u}$ is the vector of frequency coefficients.

There are many different choices possible for $\bar{x}_{df}(t)$. However, the most useful functions for discrete derivatives are purely real-valued interpolation polynomials (i.e. they don't result in complex values) [67, 68]. The reason there are many choices possible for $\bar{x}_{df}(t)$ is aliasing: any term $\tilde{u}_p e^{j\frac{2\pi}{M}lp}$ in Equation (A.60) can be replaced by $\tilde{u}_p e^{j\frac{2\pi}{M}l(p+m_pM)}$, for any integer m_p and still give the same samples $\bar{x}_l$ [67]. The addition of the m_p term means that $\bar{x}_{df}(t)$ oscillates m_p extra times between sample points.

The interpolation is then found by replacing $\tilde{u}_p e^{j\frac{2\pi}{M}lp}$ with $\tilde{u}_p e^{j\frac{2\pi}{M}l(p+m_pM)}$ and substituting $l = \frac{Mt}{T}$ into Equation (A.60) and allowing aliasing with any integer value for m_p [67]:

$$\bar{x}_{df}(t) = \sum_{p=0}^{M-1} \tilde{u}_p e^{j\frac{2\pi}{T}(p+m_pM)t}, \tag{A.63}$$

where m_p is known as the aliasing integer and is selected to minimize the mean-square slope of $\bar{x}_{df}(t)$ for $0 \le p \le M-1$. The mean-square slope is given as[I]:

$$\frac{1}{T}\int_0^L \left|\bar{x}'_{df}(t)\right|^2 dt = \frac{1}{T}\int_0^L \left|\sum_{p=0}^{M-1} j\frac{2\pi}{T}(p+m_pM)\tilde{u}_p e^{j\frac{2\pi}{T}(p+m_pM)t}\right|^2 dt \tag{A.64}$$

$$= \left(\frac{2\pi}{T}\right)^2 \sum_{p=0}^{M-1} \left|\tilde{u}_p\right|^2 (p+m_pM)^2. \tag{A.65}$$

[I]For more details on this formula, please see [67]

Thus, selecting m_p to minimize $(p + m_p M)^2$ will minimize the mean-square slope given by Equation (A.65). If $0 \leq p < \frac{M}{2}$, then $(p + m_p M)^2$ is minimized when $m_p = 0$. If $\frac{M}{2} < p < M$, then $(p + m_p M)^2$ is minimized when $m_p = -1$. When $p = \frac{M}{2}$, $m_p = 0$ and $m_p = -1$ result in the same value for $(p + m_p M)^2$. This is addressed by evenly splitting the $\tilde{u}_{\frac{1}{2}}$ term between $m_p = 0$ and $m_p = -1$. The final interpolating polynomial is given by:

$$\bar{x}_{df}(t) = \sum_{p=0}^{\frac{M}{2}-1} \tilde{u}_p e^{j\frac{2\pi}{T}pt} + \sum_{p=\frac{M}{2}+1}^{M} \tilde{u}_p e^{+j\frac{2\pi}{T}(p-M)t} + \tilde{u}_{\frac{M}{2}} \left(\frac{e^{j\frac{\pi M}{T}t} + e^{-j\frac{\pi M}{T}t}}{2} \right). \tag{A.66}$$

The derivative of $\bar{x}_{df}(t)$ is given by:

$$\begin{aligned} \bar{x}'_{df}(t) &= \sum_{p=0}^{\frac{M}{2}-1} j\frac{2\pi}{T} p\tilde{u}_p e^{j\frac{2\pi}{T}pt} - \sum_{p=\frac{M}{2}+1}^{M} j\frac{2\pi}{T} p\tilde{u}_p e^{-j\frac{2\pi}{T}(p-M)t} + j\frac{\pi M}{T}\tilde{u}_{\frac{M}{2}} \left(\frac{e^{j\frac{\pi M}{T}t} - e^{-j\frac{\pi M}{T}t}}{2} \right). & \text{(A.67)} \\ &= \sum_{p=0}^{\frac{M}{2}-1} j\frac{2\pi}{T} p \left(\tilde{u}_p e^{j\frac{2\pi}{T}pt} - \tilde{u}_{M-p} e^{-j\frac{2\pi}{T}pt} \right) - \frac{\pi M}{T}\tilde{u}_{\frac{M}{2}} sin\left(\frac{\pi M}{T} t \right). & \text{(A.68)} \end{aligned}$$

The derivatives of each of each sample of $\bar{x}$ is then estimated by evaluating $\bar{x}'_{df}(t)$ at the sample points $t = \frac{lT}{M}$[I]:

$$\bar{x}'_l = \bar{x}'_{df,l}(\frac{lT}{M}) = \sum_{p=0}^{\frac{M}{2}-1} j\frac{2\pi}{T} p \left(\tilde{u}_p e^{j\frac{2\pi}{M}pl} - \tilde{u}_{M-p} e^{-j\frac{2\pi}{M}pl} \right) = \sum_{p=0}^{M-1} \tilde{u}'_p e^{j\frac{2\pi}{M}pl} \tag{A.69}$$

In [67], it is shown that the derivative operation can be performed using the discrete Fourier Transform by multiplying each frequency domain coefficient by:

$$d_p = \begin{cases} \frac{j2\pi p}{T} & p < \frac{M}{2} \\ \frac{j2\pi}{T}(p - M) & p > \frac{M}{2} \\ 0 & p = \frac{M}{2} \text{ (if } M \text{ is even)} \end{cases} . \tag{A.70}$$

To perform the derivatives in the frequency domain, the time domain waveform is first transformed into the frequency domain using $\mathcal{F}$, the derivative is applied using d_p, and then the result is transformed back into the time domain with $\mathcal{F}^{-1}$. This operation can be performed by using a matrix, D_f, that has the form:

$$D_f = \mathcal{F}^{-1} \frac{j2\pi}{T} \begin{bmatrix} 0 & 1 & 2 & \cdots & \frac{M}{2}-1 & 0 & \frac{M}{2}+1 & \frac{M}{2}+2 & \cdots & \frac{M}{2}+\frac{M}{2}-1 & M \\ 0 & 1 & 2 & \cdots & \frac{M}{2}-1 & 0 & \frac{M}{2}+1 & \frac{M}{2}+2 & \cdots & \frac{M}{2}+\frac{M}{2}-1 & M \\ 0 & 1 & 2 & \cdots & \frac{M}{2}-1 & 0 & \frac{M}{2}+1 & \frac{M}{2}+2 & \cdots & \frac{M}{2}+\frac{M}{2}-1 & M \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & 2 & \cdots & \frac{M}{2}-1 & 0 & \frac{M}{2}+1 & \frac{M}{2}+2 & \cdots & \frac{M}{2}+\frac{M}{2}-1 & M \end{bmatrix} \mathcal{F}. \tag{A.71}$$

[I]The $\frac{\pi M}{T}\tilde{u}_{\frac{M}{2}} sin\left(\frac{\pi M}{T}t\right)$ term vanishes because $sin(\pi l) = 0$ for all l.

Appendix B

Details of Derivation of Equation 5.20

Since the order of the coefficients does not affect the result of calculating the square norm, Equation (5.20) can be written as:

$$\|\hat{\boldsymbol{d}}_\Delta\|_2 = \|\boldsymbol{P}_r(-\hat{\boldsymbol{f}}(\hat{\boldsymbol{v}}^j) - \hat{\boldsymbol{J}}_\Delta \Delta\hat{\boldsymbol{v}}_g)\|_2. \tag{B.1}$$

Using Equations (5.17) and (5.18), Equation (B.1) can be written as:

$$\|\hat{\boldsymbol{d}}_\Delta\|_2 = \| \begin{bmatrix} \hat{\boldsymbol{f}}_1 \\ \hat{\boldsymbol{f}}_2 \end{bmatrix} - \boldsymbol{P}_r \hat{\boldsymbol{J}}_\Delta \boldsymbol{P}_c \boldsymbol{U}^{-1} \hat{\boldsymbol{y}}_1 \|_2. \tag{B.2}$$

Substituting Equation (5.15) into Equation (B.2), and partitioning $\boldsymbol{L}$ as in Equation (5.17) results in:

$$\|\hat{\boldsymbol{d}}_\Delta\|_2 = \| \begin{bmatrix} \hat{\boldsymbol{f}}_1 \\ \hat{\boldsymbol{f}}_2 \end{bmatrix} - \begin{bmatrix} \boldsymbol{L}_1 \\ \boldsymbol{L}_2 \end{bmatrix} \hat{\boldsymbol{y}}_1 \|_2. \tag{B.3}$$

Since $\boldsymbol{L}_1\hat{\boldsymbol{y}}_1 = \hat{\boldsymbol{f}}_1$ and $\hat{\boldsymbol{d}}_\Delta = \hat{\boldsymbol{f}}_2 - \boldsymbol{L}_2\hat{\boldsymbol{y}}_1$:

$$\|\hat{\boldsymbol{d}}_\Delta\|_2 = \| \begin{bmatrix} \hat{\mathbf{0}}_1 \\ \hat{\boldsymbol{d}}_\Delta \end{bmatrix} \|_2, \tag{B.4}$$

where $\hat{\mathbf{0}}_1$ is a vector with m entries with all entries set to zero.

Bibliography

[1] L. Nagel, "The life of spice," *transcript of a presentation given at BCTM'96 and hosted by The Designer's Guide Community*, January 2008.

[2] L. Nagel and R. Rohrer, "Computer analysis of nonlinear circuits, excluding radiation (cancer)," *IEEE Journal of Solid-State Circuits*, vol. 6, no. 4, pp. 166–182, 1971.

[3] T. Perry, "Donald o. pederson [electronic engineering biography]," *IEEE Spectrum*, vol. 35, no. 6, pp. 22–27, 1998.

[4] Ognen Nastov, Rircardo Telichevesky, Ken Kundert, and Jacob White, "Fundamentals of Fast Simulation Algorithms for RF Circuits," *Proceedings of the IEEE*, vol. 95, pp. 600–621, March 2007.

[5] Kenneth S. Kundert, "Simulation Methods for RF Integrated Circuits," *ICCAD-97 IEEE International Conference on Computer-Aided Design*, pp. 752–765, November 1997.

[6] Kenneth S. Kundert and Alberto Sangiovanni-Vincentelli, "Finding the Steady-State Response of Analog and Microwave Circuits," *CICC-88 Proceedings of the 1988 IEEE Custom Integrated Circuits Conference*, pp. 611–617, May 1988.

[7] Vittorio Rizzoli and Antrea Neri, "State of the Art and Present Trends in Nonlinear Microwave CAD Techniques," *IEEE Transactions on Microwave Theory and Techniques*, vol. 36, pp. 343–365, February 1988.

[8] Kai Bittner and Hans Georg Brachtendorf, "Fast Algorithms for Adaptive Free-Knot Spline Approximation Using Non-Uniform Biorthogonal Spline Wavelets," *SIAM Journal on Scientific Computing*, vol. 37, pp. B283–B304, March 2015.

[9] K. Bittner and E. Dautbegovic, "Adaptive wavelet-based method for simulation of electronic circuits," in *Scientific Computing in Electrical Engineering SCEE 2010*, pp. 321–328, Springer Berlin Heidelberg, Oct. 2011.

[10] Kai Bittner and Emira Dautbegovic, "Wavelet Algorithm for Circuit Simulation," *COMPEL International Journal of Computations and Mathematics in Electrical*, April 2016.

[11] Kai Bittner and Hans Georg Brachtendorf, "Adaptive Multi-Rate Wavelet Method for Circuit Simulation," *Radioeng*, vol. 23, pp. 300–307, April 2014.

[12] Akira Ohkubo, Seiichiro Moro, and Tadashi Matsumoto, "A Method for Circuit Analysis using Haar Wavelet Transform," *The 47th IEEE International Midwest Symposium on Circuits and Systems*, vol. 3, pp. 399–402, July 2004.

[13] Masanori Oishi, Seiichiro Moro, and Tadashi Matsumoto, "A Modified Method for Circuit Analysis using Haar Wavelet Transform with Adaptive Resolution -For Circuits with Waveform with Sharp Convex Ranges-," *ECCTD 2009 European Conference on Circuit Theory and Design*, pp. 299–302, August 2009.

[14] M. Liu, C. K. Tse, and J. Wu, "A wavelet approach to fast approximation of steady-state waveforms of power electronics circuits," *International Journal of Circuit Theory and Applications*, vol. 31, no. 6, pp. 591–610, 2003.

[15] K. Tam, S.-C. Wong, and C. Tse, "An improved wavelet approach for finding steady-state waveforms of power electronics circuits using discrete convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 52, no. 10, pp. 690–694, 2005.

[16] Nick Soveiko and Michel Nakhla, "Wavelet Harmonic Balance," *IEEE Microwave and Wireless Components Letters*, vol. 13, pp. 232–234, June 2003.

[17] Nick Soveiko and Michel Nakhla, "On the Steady-State Analysis of Nonlinear Circuits With Frequency Dependent Parameters in Wavelet Domain," *IEEE Microwave and Wireless Components Letters*, vol. 15, pp. 384–386, May 2005.

[18] K. Fedick and C. Christoffersen, "Effect of wavelet selection on periodic steady-state analysis," *IEEE Access*, vol. 8, pp. 70784–70796, 2020.

[19] Brij N. Singh and Arbind K. Tiwari, "Optimal Selection of Wavelet Basis Function Applied to ECG Signal Denoising," *Digital Signal Processing*, vol. 16, pp. 275–287, May 2006.

[20] Walid G. Morsi and M. E. El-Hawary, "The Most Suitable Mother Wavelet For Steady-State Power System Distorted Waveforms," in *Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 17–22, IEEE, May 2008.

[21] X. Ma, C. Zhou, and I. J. Kemp, "Automated Wavelet Selection and Thresholding for PD Detection," *IEEE Electrical Insulation Magazine*, vol. 18, pp. 37–45, March/April 2002.

[22] G. Keppel and S. Zedeck, *Data Analysis for Research Designs: Analysis of Variance and Multiple Regression/Correlation Approaches*. New York: W. H. Freeman, 1989.

[23] Jing Gao, "Comparison Analysis Based on the Cubic Spline Wavelet and Daubechies Wavelet of Harmonic Balance Method," *Abstract and Applied Analysis*, vol. 2014, 2014. Article ID 634974.

[24] Jirí Vlach and Kishore Singhal, *Computer Methods for Circuit Analysis and Design*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1983. ISBN: 978-0442281083.

[25] Paulo J. C. Rodrigues, *Computer-Aided Analysis of Nonlinear Microwave Circuits*. Norwood, MA: Artech House, 1998. ISBN: 0-89006-690-6.

[26] Stephen A. Maas, *Nonlinear Microwave and RF Circuits*. Norwood, MA: Artech House, 1997. ISBN: 1-58053-484-8.

[27] Bryan T. Morrison, "An Introduction to Gyrator Theory: How inductors can be simulated using resistors, capacitros, and opamps," *Popular Electronics*, pp. 58–59, July 1977.

[28] Thomas H. Lynch, "The right gyrator trims the fat off active filters," *Electronics*, pp. 115–119, July 1977.

[29] C. Christoffersen, "Cardoon circuit simulator." Online: https://vision.lakeheadu.ca/cardoon/, accesed: October 2020.

[30] J. F. Oliveira and J. C. Pedro, "A new time-domain simulation method for highly heterogeneous RF circuits," *Proceedings of the 37th European Microwave Conference (EUMC '07)*, pp. 1161–1164, 2007.

[31] Jorge F. Oliveira and José Carlos Pedro, "An Efficient Time-Domain Simulation Method for Multirate RF Nonlinear Circuits," *IEEE Transactions on Microwave Theory and Techniques*, vol. 55, pp. 2384–2392, November 2007.

[32] Ricardo Telichevesky, Ken Kundert, and Jacob White, "Efficient AC and Noise Analysis of Two-Tone RF Circuits," *DAC '96 Proceedings of the 33rd annual Design Automation Conference*, pp. 292–297, June 1996.

[33] Dan Feng, Joel Phillips, Keith Nabors, Ken Kundert, and Jacob White, "Efficient Computation of Quasi-Periodic Circuit Operating Conditions via a Mixed Frequency/Time Approach," *DAC-99 Proceedings of the 36th Design Automation Conference*, pp. 635–640, 1999 1999.

[34] Jaijeet Roychowdhury, "Efficient Methods for Simulating Highly Nonlinear Multi-Rate Circuits," *Proceedings of the 34th Design Automation Conference*, pp. 269–274, June 1997.

[35] John R. Williams and Kevin Amaratunga, "Indroduction to Wavelets in Engineering," *International Journal for Numerical Methods in Engineering*, vol. 37, no. 14, pp. 2365–2388, 1994.

[36] Amara Graps, "An Introduction to Wavelets," *IEEE Computational Science and Engineering*, vol. 2, pp. 50–61, June 1995.

[37] B. Vidakovic and P. Mueller, "Wavelets for kids: A tutorial introduction." Online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.70.54, 1991. accessed October 2020.

[38] Liu, Chun-Lin, "A tutorial of the wavelet transform." Online: https://www.researchgate.net/publication/265633800_A_Tutorial_of_the_Wavelet_Transform, accessed October 2020, January 2010.

[39] W. H. Press, *Numerical Recipes in Fortran 77: The Art of Scientific Computing*. Cambridge University Press, sep 1992.

[40] Stéphane Mallat with contributions from Gabriel Peyré, *A Wavelet Tour of Signal Processing: The Sparse Way*. Elsevier, 2009.

[41] A. Karoui and R. Vaillancourt, "Families of biorthogonal wavelets," *Computers & Mathematics with Applications*, vol. 28, pp. 25–39, Aug. 1994.

[42] S. S. Rao, *Applied Numerical Methods for Engineers and Scientists*, ch. Solution of Nonlinear Equations: 2.5 Newton-Raphson Method, pp. 64–70. Prentice Hall, 2002.

[43] Chapra, S.C., *Applied Numerical Methods with MATLAB for Engineers and Scientists*. McGraw-Hill International edition, McGraw-Hill Higher Education, 2005.

[44] Dongarra, Jack and Eijkhout, Victor and Luszczek, Piotr, "Recursive approach in sparse matrix lu factorization," *Scientific Programming*, vol. 9, 08 2002.

[45] Cong Fu and Xiangmin Jiao and Tao Yang, "Efficient sparse lu factorization with partial pivoting on distributed memory architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 2, pp. 109–125, 1998.

[46] Michel J. Daydé and lain S. Duff, "Level 3 blas in lu factorization on the cray-2, eta-10p, and ibm 3090-200/vf," *The International Journal of Supercomputing Applications*, vol. 3, pp. 40–70, June 1989.

[47] Davis, Timothy A. and Duff, Iain S., "An unsymmetric-pattern multifrontal method for sparse lu factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 18, no. 1, pp. 140–158, 1997.

[48] Timothy A. Davis, "A column pre-ordering strategy for the unsymmetric-pattern multifrontal method," *ACM Transactions on Mathematical Software*, vol. 30, pp. 165–195, June 2004.

[49] Timothy A. Davis, "Suitesparse : A suite of sparse matrix software." Online: https://people.engr.tamu.edu/davis/suitesparse.html, accessed October 2020.

[50] Dian Zhou and Wei Cai, "A Fast Wavelet Collocation Method for High-Speed Circuit Simulation," *IEEE Transactions on Circuits and Systems*, vol. 46, pp. 920–930, August 1999.

[51] Dian Zhou, Wei Cai, and Wu Zhang, "An Adaptive Wavelet Method for Nonlinear Circuit Simulation," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 46, pp. 931–938, August 1999.

[52] Carlos E. Christoffersen and Michael B. Steer, "Comparison of Wavelet and Time Marching-Based Microwave Circuit Transient Analyses," *2001 IEEE MTT-S International Microwave Symposium Digest*, vol. 1, pp. 447–450, May 2001.

[53] Nick Soveiko and Michel S. Nakhla, "Steady-State Analysis of Multitone Nonlinear Circuits in Wavelet Domain," *IEEE Transactions on Microwave Theory and Techniques*, vol. 52, pp. 785–797, March 2004.

[54] Kai Bittner, *Wavelets and Splines: Athens 2005*, ch. Biorthogonal Spline Wavelets on the Interval, pp. 93–104. Nashboro Press, 2006.

[55] J. Wang, "Cubic Spline Wavelet Bases of Sobolev Spaces and Multilevel Interpolation," *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 154–163, 1996.

[56] Matthias Bucher, Christophe Lallement, Christian Enz, Fabien Theodoloz, Francois Krummenacher, "The EPFL-EKV MOSFET Model Equations for Simulation, Technical Report, Model Version 2.6," *Technical Report, EPFL*, July 1998.

[57] C. Christoffersen, T. Ngo, R. Song, Y. Zhou, S. Pichardo, and L. Curiel, "Suboptimal Class DE Operation for Ultrasound Transducer Arrays," *Proceedings of the 2018 IEEE NEWCAS conference, Montreal*, pp. 1–4, June 2018.

[58] Márcio Cherem Schneider and Carlos Galup-Montoro, "An MOS Transistor Model for Analog Circuit Design," *IEEE Journal of Solid-State Circuits*, vol. 33, pp. 1510–1519, October 1998.

[59] S. N. Lalgudi, E. Engin, G. Casinovi, and M. Swaminathan, "Accurate transient simulation of interconnects characterized by band-limited data with propagation delay enforcement in a modified nodal analysis framework," *IEEE Transactions on Electromagnetic Compatibility*, vol. 50, pp. 715–729, 2008.

[60] Sonali Luniya et al., "Compact electrothermal modeling of an x-band mmic," *2006 IEEE MTT-S International Microwave Symposium Digest*, pp. 651–654, June 2006.

[61] T. Blumensath, M. E. Davies, and G. Rilling, "Greedy algorithms for compressed sensing," in *Compressed Sensing: Theory and Applications* (Y. C. Eldar and G. Kutyniok, eds.), pp. 348–393, Cambridge University Press, May 2012.

[62] G. Peters and J. H. Wilkinson, "The least squares problem and pseudo-inverses," *The Computer Journal*, vol. 13, pp. 309–316, January 1970.

[63] Å. Björck and I. S. Duff, "A direct method for the solution of sparse linear least squares problems," *Linear Algebra and its Applications*, vol. 34, pp. 43–67, December 1980.

[64] M. Heath, "Numerical methods for large sparse linear least squares problems," *SIAM Journal on Scientific and Statistical Computing*, vol. 5, pp. 497–513, September 1984.

[65] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics, Jan. 1994.

[66] Hossain, Md, "ECG Signal Compression using Energy Compaction Based Thresholding of the Wavelet Coefficients," *Duet Journal*, vol. 1, July 2011.

[67] S. G. Johnson, "Notes on fft-based differentiation." Online: https://math.mit.edu/ stevenj/fft-deriv.pdf, accessed: October 2020 2011.

[68] A. N. Malyshev, "On the spectral differentiation." Online: http://www.ii.uib.no/ sasha/INF263/spectral, accessed: October 2020.

[69] Dennis Roddy and John Coolen, *Electronic Communications 4th Edition*. Prentice Hall, 1995.