

Physical Modelling meets Machine Learning: Performing Music with a Virtual String Ensemble

Graham Keith Percival

Submitted in the fulfilment of the requirements for the
Degree of Doctor of Philosophy

School of Engineering
College of Engineering and Physical Sciences
University of Glasgow

May 2013

Abstract

This dissertation describes a new method of computer performance of bowed string instruments (violin, viola, cello) using physical simulations and intelligent feedback control. Computer synthesis of music performed by bowed string instruments is a challenging problem. Unlike instruments whose notes originate with a single discrete excitation (e.g., piano, guitar, drum), bowed string instruments are controlled with a continuous stream of excitations (i.e. the bow scraping against the string). Most existing synthesis methods utilize recorded audio samples, which perform quite well for single-excitation instruments but not continuous-excitation instruments.

This work improves the realism of synthesis of violin, viola, and cello sound by generating audio through modelling the physical behaviour of the instruments. A string's wave equation is decomposed into 40 modes of vibration, which can be acted upon by three forms of external force: A bow scraping against the string, a left-hand finger pressing down, and/or a right-hand finger plucking. The vibration of each string exerts force against the instrument bridge; these forces are summed and convolved with the instrument body impulse response to create the final audio output. In addition, right-hand haptic output is created from the force of the bow against the string. Physical constants from ten real instruments (five violins, two violas, and three cellos) were measured and used in these simulations. The physical modelling was implemented in a high-performance library capable of simulating audio on a desktop computer one hundred times faster than real-time. The program also generates animated video of the instruments being performed.

To perform music with the physical models, a virtual musician interprets the musical score and generates actions which are then fed into the physical model. The resulting audio and haptic signals are examined with a support vector machine, which adjusts the bow force in order to establish and maintain a good timbre. This intelligent feedback control is trained with human input, but after the initial training is completed the virtual musician performs autonomously. A PID controller is used to adjust the position of the left-hand finger to correct any flaws in the pitch. Some performance parameters (initial bow force, force correction, and lifting factors) require an initial value for each string and musical dynamic; these are calibrated automatically using the previously-trained support vector machines. The timbre judgements are retained after each performance and are used to preemptively adjust bowing parameters to avoid or mitigate problematic timbre for future performances of the same music.

The system is capable of playing sheet music with approximately the same ability level as a human music student after two years of training. Due to the number of instruments measured and the generality of the machine learning, music can be performed with ensembles of up to ten stringed instruments, each with a distinct timbre. This provides a baseline for future work in computer control and expressive music performance of virtual bowed string instruments.

Dedication

This work is dedicated to human creativity everywhere, be it musicians writing documentation for open-source software, programmers creating music videos to accompany computer-generated singing, doctors playing in orchestras, or academics writing fanfiction. The desire to be creative outside of professional commitments is one of humanity's greatest qualities.

Contents

Abstract	i
Dedication	ii
List of Tables	vii
List of Figures	x
List of Accompanying Material	xi
List of Variables	xiii
Preface	xv
Acknowledgements	xvi
Author's Declaration	xvii
1 Introduction	1
1.1 Motivation and applications	2
1.1.1 Musical robotics	2
1.1.2 Computer singing synthesis	3
1.1.3 Accessibility and assistive technologies	5
1.1.4 Audio synthesis for composers, performers, and musicologists	6
1.1.5 Free / open source software, copyleft art, and open access	7
1.1.6 Research constraints to generalize applicability	9
1.2 Literature review	9
1.2.1 Violin physics and mechanical control	9
1.2.2 Violin physical modelling	13
1.2.3 Alternate violin synthesis methods	14
1.2.4 Control of musical synthesis and instruments	16
1.2.5 Music information retrieval	18
1.2.6 Music education and expressive performance	21
1.2.7 Programming and implementation	22
1.3 Problem definition revisited	23
1.3.1 Overview of <i>Vivi, the Virtual Violinist</i>	24
1.3.2 Choice of violin physical model	25
1.4 Organization of this dissertation	26

I Physical Modelling of the Violin Family	28
2 Physics of Physical Modelling	29
2.1 String and instrument body simulation	31
2.1.1 String physics	32
2.1.2 Finger forces on the string	34
2.1.3 Bow force on the string	39
2.1.4 String states	45
2.1.5 Instrument body simulation	46
2.2 Design decisions and consequences for finger actions	48
2.2.1 Damped spring action	48
2.2.2 Two forces for right-hand plucking	49
2.2.3 Two forces for the left-hand finger	50
2.2.4 Three finger forces in total	55
2.3 Design decisions and consequences for bowing actions	56
2.3.1 “Strongly rational” positions in bowing	57
2.3.2 String moving faster than the bow	58
2.3.3 “Safety valve” changes of slip-state	62
2.4 Final remarks on the physics	64
2.4.1 Model simplifications	64
2.4.2 Summary of available actions	66
3 Constants for Physical Modelling	67
3.1 Overview of instruments and strings	68
3.2 String modal decays	70
3.2.1 Experimental procedure and analysis techniques	71
3.2.2 String inharmonicity and non-linearity	73
3.2.3 Detecting modal decays	77
3.3 Instrument body impulse responses	82
3.3.1 Experimental procedure and analysis techniques	82
3.3.2 Instrument body impulse responses	83
3.4 Estimating remaining constants	84
3.4.1 Bounds of physical constants in the literature	84
3.4.2 Fitting unknown string constants to measured constants	85
3.5 Constants selected from simulations	86
3.5.1 Number of modes	86
3.5.2 String sampling rates	88
3.5.3 Finger damping and plucking constants	90
3.5.4 Tuning string tension	91
3.5.5 Calculating cut-off displacement and velocity	93
3.6 Final remarks on the constants	94
3.6.1 More accurate measurements	94
3.6.2 Final constants used	96

4 Implementation of Physical Modelling	97
4.1 Video generation	99
4.1.1 Blender models	99
4.1.2 Blender animation	100
4.2 Implementation notes for physical modelling	101
4.2.1 Discrete-time signals and music	101
4.2.2 Vectorized optimizations	102
4.2.3 Subnormal numbers	103
4.3 Artifastring: Artificial Fast String	104
4.3.1 C++ implementation	104
4.3.2 Benchmarks, profiling, and future improvements	108
4.3.3 Artifastring programs	111
4.3.4 Realtime interactive use of Artifastring	112
4.4 Final remarks on the virtual violin family	113
4.4.1 The model in general	113
4.4.2 Public interface for <code>ArtifastringInstrument</code>	115
II Performing with the Virtual Violin Family	116
5 Control loops	117
5.1 General approach to the virtual violinist	118
5.1.1 Non-deterministic system	118
5.1.2 Control theory and human control of violins	119
5.1.3 Pedagogical inspiration	121
5.2 Right hand: Sound quality and bow force F_b	123
5.2.1 Human input: Classifying sound quality	124
5.2.2 Feature extraction and Support Vector Machine classification	125
5.2.3 Bow control loop	128
5.3 Left hand: Pitch and finger position x_g	130
5.4 Final remarks on the control loops	132
5.4.1 Plucked notes	132
5.4.2 Limitations of the bow control	133
5.4.3 Extending the control loops	135
5.4.4 Mid-level note modifiers	138
6 Calibration, Performance, and Self-Improvement	139
6.1 Training and calibration	140
6.1.1 Interactive training	140
6.1.2 Accuracy of the SVMs	141
6.1.3 Verifying the SVMs	141
6.1.4 Initial bow force F_b^i and correction factor K	142
6.1.5 Bow lifting factor D_b	145
6.1.6 Musical exercises	145

6.2	Interpretation of musical notation	147
6.2.1	Default note handling and pitch	147
6.2.2	Notation with fixed meaning	148
6.2.3	Notation with ambiguous meaning	149
6.3	Practicing a piece of music	150
6.3.1	Generating hill-climbing candidates	150
6.3.2	Choosing the steps	151
6.4	Final remarks on calibration, music, and self-improvement	152
6.4.1	Improvements to the calibration and practicing	152
6.4.2	Improvements to the musical interpretation	154
7	Implementation of <i>Vivi, the Virtual Violinist</i>	156
7.1	Extracting events from notation	158
7.1.1	Extracting notation events from LilyPond	158
7.1.2	Clickable scores	160
7.1.3	Extracting notation from other software	160
7.2	Additional features for practical use	161
7.2.1	Scores with multiple instruments	161
7.2.2	Reducing skill level	162
7.2.3	Chorusing effect	163
7.3	Design of <i>Vivi</i>	164
7.3.1	Languages and libraries used	164
7.3.2	Parallel processing: Queue, dispatcher, workers, and multi-stage jobs	164
7.3.3	Benchmarks and profiling	166
7.3.4	Realtime interactive use of <i>Vivi</i>	167
7.4	Final remarks on <i>Vivi, the Virtual Violinist</i>	169
7.4.1	Comparison of <i>Vivi</i> with alternate performance methods	169
7.4.2	Future work on <i>Vivi</i>	174
8	Conclusion	176
8.1	Evaluating the initial goal	176
8.2	Philosophical implications (or the lack thereof)	178
8.3	Contributions of this dissertation	179
8.4	Future work	180
A	Additional Mathematics for Physical Modelling	181
B	Performances of select sheet music	185
C	Source code, raw data, and copyleft licenses	188
	Bibliography	189

List of Tables

2.1	Main variables used in bowed-string algorithm.	30
2.2	Coefficients for modal displacement and velocity.	33
2.3	Coefficients for pluck release.	36
2.4	Coefficients for bow force on the string.	40
2.5	Unit string with simple physical constants.	50
2.6	Limits on v_0^h for transition to a sticking slip-state.	60
3.1	Overview of instruments tested	69
3.2	Measured string lengths and diameters	69
3.3	Fundamental frequencies f_0 and inharmonicity B for measured strings.	76
3.4	Removing suspicious partials from decay-fitting.	79
3.5	Fitted modal decays of measured strings.	81
3.6	Bounds of string constants used in the bowed-string algorithm.	84
3.7	Bounds of bowing constants used in the bowed-string algorithm.	84
3.8	String frequency multipliers.	89
3.9	Equal-loudness plucking force.	94
3.10	Final constants used	96
4.1	Class variables of <code>ArtifastringString</code>	105
4.2	Benchmarks on common CPUs	108
4.3	Profiling Artifastring	109
4.4	API for <code>ArtifastringInstrument</code>	115
5.1	Physical parameters determined by dynamic.	123
5.2	Human judgements of bowing	123
6.1	Number of files labelled in the datasets	145
7.1	Default panning for multi-instrument ensembles	162
7.2	Benchmarks of <i>Vivi</i> on common CPUs	167
7.3	Profiling <i>Vivi</i>	167

List of Figures

1.1	Sheet music to video with no human interaction.	1
1.2	Overview of music performance with <i>Vivi</i> .	24
1.3	Sampling rates of various parts of <i>Vivi</i> .	26
2.1	Physical modelling in context.	29
2.2	Overview of the violin physical modelling.	30
2.3	Overview of the violin string physical modelling.	31
2.4	String displacements at beginning and release of pluck on the violin G string.	35
2.5	Finger forces F_0 and F_1 causing desired pitch after a pluck release.	37
2.6	String displacements during one cycle of a pluck on the violin G string.	38
2.7	Hyperbolic friction curve.	40
2.8	Bow slipping states.	42
2.9	Movement of a bowed string	43
2.10	Bow harmonics.	44
2.11	String states.	45
2.12	Frequency response of violin instrument body filters.	47
2.13	Two plucks with damped and undamped plucking springs.	48
2.14	Comparison of one and two plucking forces.	49
2.15	Comparison of one and two finger forces during plucking.	50
2.16	Beating in the unit string.	51
2.17	Modal displacements during beating in the unit string.	52
2.18	System behaviour of modes 1 and 3 in the unit string	54
2.19	Comparison of varying numbers of forces for plucks.	55
2.20	Frequency bands of bowing position x_b analysis.	56
2.21	Effect of rational bow positions x_b on violin behaviour.	57
2.22	Low sampling rate causing the string to move faster than the bow.	59
2.23	Friction slip-states transitions from a negative slip-state.	60
2.24	Friction slip-states for a bowing change.	62
2.25	Effect of forbidding positive slips.	63
2.26	Effect of forbidding skipping over the stick-state.	63
3.1	Physical modelling in context.	67
3.2	Spectrogram of plucking a violin E, viola D, and cello C string, first four modes.	70
3.3	Photos of experimental setup for measuring decay of string velocity.	71
3.4	Pluck decay (first window) and noise of violin E and cello C strings.	72
3.5	Non-linearities in the violin E, viola D, and cello C string.	74

3.6	Detecting string inharmonicity.	75
3.7	f_0 and B fits for the violin E, viola D, and cello C strings.	77
3.8	Selected partials of violin E, viola D, and cello D, one pluck.	78
3.9	Detected and fitted decays for violin E, viola D, and cello C strings.	80
3.10	Photos of the pendulum for tapping the bridge.	82
3.11	Time-domain impulse response of a violin and a cello with a standard tap.	83
3.12	Frequency impulse responses of all instrument bodies.	83
3.13	Different modes N for the plucked and bowed violin E and cello C strings.	87
3.14	Testing parameters for frequency multipliers M_f	88
3.15	Audio quality resulting from frequency multipliers M_f , all strings.	89
3.16	RMS decay for violin G string with open plucks and fingered plucks	90
3.17	RMS and SFM during the “pull” portion of a pluck.	91
3.18	Relationship of tension to pitch, violin A string.	92
3.19	Pitch-tuning tension control loop.	92
3.20	RMS output for instrument body and violin E string.	93
4.1	Context of this chapter.	97
4.2	Benchmark test for physical actions	98
4.3	Blender models of a violin	99
4.4	Music excerpt for buffer timing problems.	101
4.5	Class diagram for the Artifastring library.	105
4.6	C++ code to process a pluck	106
4.7	Using Artifastring in C++ and python	107
4.8	Beginning of <code>twinkle.notes</code> used with <code>note2actions.py</code>	111
4.9	Interactive control of Artifastring with a tablet	112
5.1	Intelligent feedback control in context.	117
5.2	Different output with identical bowing parameters	118
5.3	“Basic fingers” evaluated for training and calibration	124
5.4	Bow force judgements with only two features	127
5.5	Comparison of direct SVM output and weighted probabilities	127
5.6	Bow force F_b control loop.	128
5.7	Two examples of bow control, violin D string f	129
5.8	Tuning finger position control loop.	130
5.9	Example of pitch control	131
5.10	Example of pitch glissando control	131
5.11	Problematic control of the cello C string	133
5.12	Low-level note modifiers	138
6.1	Calibration, performance, and self-improvement in context	139
6.2	Interactive training with a calibration exercise	140
6.3	Interactive training with cello sheet music	140
6.4	Checking the output of the SVMs	141
6.5	Simulated notes for bow force calibration	143

6.6	Calibrating F_b^i and K	143
6.7	Calibrated F_b^i	144
6.8	Calibrating bow lifting factor D_b	145
6.9	Musical exercises for <i>Vivi</i>	146
6.10	Sample of LilyPond input format with generated output	147
6.11	String and finger positions	147
6.12	Slurs, ties, <i>pizzicato</i> , and <i>arco</i>	148
6.13	Musical notation with no fixed meaning	149
6.14	Practice with hill climbing	151
7.1	Context of this chapter.	156
7.2	Screenshots of <i>Vivi</i>	157
7.3	LilyPond input, graphical output, and extracted music events	158
7.4	Excerpts of <code>event-listener.ly</code>	159
7.5	Bach double violin concerto with continuo	161
7.6	Scale in thirds with two violins	162
7.7	Scale arranged for chorus of all instruments	163
7.8	Design diagram of <i>Vivi</i>	165
7.9	Black-box file for testing, benchmarks, and profiling	166
7.10	Interactive use of force correction with a tablet	168
7.11	Comparison of <i>Vivi</i> and MIDI	170
7.12	Comparison of <i>Vivi</i> and human musicians playing “Twinkle, twinkle, little star”	172
7.13	Examination of first note of “Twinkle, twinkle, little star”	173
7.14	Frequency response of viola-II silence before impulse	173
A.1	X_{3n} and Y_{3n} for violin-E-I and cello-C-I.	184
A.2	$A_{11}B_{00} - A_{10}B_{01}$ for violin-E-I and cello-C-I.	184

List of Accompanying Material

Audio examples

2.1	Plucking a fingered violin G string	37
2.2	Normal bowed fingered string.	43
2.3	Natural harmonic.	44
2.4	Normal Helmholtz vibrations.	44
2.5	Two plucks, undamped	48
2.6	Two plucks, damped	48
2.7	Plucking an open string with one pluck force.	49
2.8	Plucking an open string with two pluck force.	49
2.9	Pluck with one finger force.	50
2.10	Pluck with two finger forces.	50
2.11	Beating in the plucked unit string.	51
2.12	Pluck with three forces, release with two forces.	55
2.13	Pluck with four forces, release with two forces.	55
2.14	Pluck with three forces, release with three forces.	55
2.15	Bowing at $x_b = 0.160L$, $x_b = 0.167L$, and $x_b = 0.172L$	56
2.16	Positive bow slipping at $f_s = 55000$ Hz.	59
2.17	No positive bow slipping at $f_s = 66150$ Hz.	59
2.18	No positive bow slipping at $f_s = 110000$ Hz.	59
2.19	Change of bowing parameters with positive slips allowed.	63
2.20	Change of bowing parameters with positive slips forbidden.	63
2.21	Change of bowing parameters with skipping over stick-state.	63
2.22	Change of bowing parameters forbidding skipping over stick-state. Caution: very loud audio at 1 second.	63
3.1	Signal of current induced from violin E, viola D, and cello C strings.	70
3.2	Violin E plucks with varying number of modes N	87
3.3	Violin E bowed notes with varying number of modes N	87
3.4	Cello C plucks with varying number of modes N	87
3.5	Cello C bowed notes with varying number of modes N	87
3.6	Examples of two plucks, $R_p = [0.2, 1.0, 10.0]$	91
3.7	Pitch-tuning tension control loop in action.	92
4.1	Audio generated from <code>benchmark.actions</code>	98
4.2	Audio generated from <code>twinkle.notes</code>	111

5.1	Two bowing simulations with identical parameters	118
5.2	Examples of bow force classifications	123
5.3	Two examples of bow control	129
5.4	Pitch correction	131
5.5	Glissando control	131
5.6	Problematic control of the cello C string	133
6.1	Interactive training of a violin calibration exercise	140
6.2	Interactive training with cello sheet music	140
6.3	Violin musical exercises	146
6.4	Viola musical exercises	146
6.5	Cello musical exercises	146
6.6	Examples of solo practice	151
7.1	Bach double violin concerto introduction	161
7.2	Three different skill levels of <i>Vivi</i>	162
7.3	Scale arranged as a trio; performed by solo instruments vs a chorus of ten instruments.	163
7.4	Black-box testing file, violin and cello.	166
7.5	Scale arranged as a chorus, MIDI and <i>Vivi</i> performances	170
7.6	Bach's Double Violin concerto, MIDI and <i>Vivi</i> performances	170
7.7	Mozart's Eine kleine Nachtmusik, MIDI and <i>Vivi</i> performances	170
7.8	Comparison of <i>Vivi</i> , a human student, and a human teacher	172
7.9	Comparison of one note of "twinkle" between teacher and <i>Vivi</i>	173
B.1	Eine kleine Nachtmusik	185
B.2	Bach double	186
B.3	Pachelbel	187
B.4	Gavotte	187

Video examples

1.1	Black box performance	1
2.1	Displacements after plucking a fingered violin G string: 4.5 ms, full sample rate.	38
2.2	Displacements after plucking a fingered violin G string: 200 ms, reduced sample rate.	38
2.3	Normal bowed fingered string, beginning.	43
2.4	Normal bowed fingered string, middle.	43
2.5	Natural harmonic	44
2.6	Normal Helmholtz vibrations.	44
4.1	Three quality levels of animation	99
4.2	Interactive control of Artifastring with a tablet	112
7.1	Bach double violin concerto introduction, video quality 0 and 1	161
7.2	Interactive control of <i>Vivi</i> with a tablet	168

List of Variables

Cast of variables in order of their appearance. SI units are assumed.

Physics of Physical Modelling

f_s	Sample rate	dt	Sampling interval
M_f	Multipler for string sample rate	s	String number
n	Mode number	N	Maximum number of modes
x_b	Bow-bridge distance	x_f	Finger-bridge distance
v_b	Bow velocity	F_b	Bow force
a_b	Bow acceleration	v_b^t	Target bow velocity
a_n	Modal displacement	\dot{a}_n	Modal velocity
F_i	External forces	D_i	Coefficients for force calculations
$A[t]$	Audio output	$H[t]$	Haptic output
ρ_L	Linear density	T	Tension
E	Young's elastic modulus	d	Diameter
L	Length	$\phi_n(x)$	Eigenvectors
$R_L(\omega)$	Frequency-dependent damping	r_n	Modal damping value
\check{f}	Modal forces	$y(x, t)$	Transverse displacement
X_{pq}	Coefficients for updating modal displacement	Y_{pq}	Coefficients for updating modal velocity
a_n^h	“Hands-free” modal displacement	\dot{a}_n^h	“Hands-free” modal velocity
y_i^h	“Hands-free” displacement at point i	v_i^h	“Hands-free” velocity at point i
A_{pq}	Modal displacement of forces at points p, q	B_{pq}	Modal velocity of forces at points p, q
K_i	Spring strength	R_i	Spring damped strength
W_p	Width of plucking finger	y_p^d	Desired plucking finger displacement
v_p	Velocity of plucking finger	t_p	Duration of plucking action
μ_s	Coefficient of static friction	μ_d	Coefficient of dynamic friction
μ_c	Coefficient of slope between static and dynamic friction	μ_e	Slightly lower (random) value of μ_c
A_{mag}	Magnitude of string vibrations in a buffer	A_{\min}	Minimum value of A_{mag} to continue processing
Δv	Relative velocity between bow and string		

Constants of Physical Modelling

f_n	Frequency of the n^{th} partial	B	String inharmonicity
$N_{\text{top}}(\omega)$	Frequency-dependent noise floor	N_{par}	Partial-specific noise floor
R^2	Coefficient of determination	B_1, B_2	Coefficients for modal damping
N_c	Maximum detectable partial number	f^r	Reference frequency of tuning
K_s	“Fudge factor” to reduce bow force while tuning	K_T	Proportional gain for tuning PID

Implementation of Physical Modelling

C_s	3-vector of bow-string contact along string	C_h	3-vector of bow-string contact along bow hair
B_I	Intersection of bow hair and bow winding	\vec{B}_A, \vec{B}_H	Orientation of bow

Control Loops

x_g	Finger distance from nut	K	Rate of change for bow force control loop
c	Classification labels (timbre judgements)	c'	Weighted classification output
F_b^i	Initial bow force	D_b	Bow lifting factor
W_a	Number of ticks for bow acceleration	W_c	Number of ticks for additional delay
P_r	Reference pitch (Hz)	M_r	Reference pitch (MIDI)
M_s	Pitch of string (MIDI)	K_M	Rate of change for finger control loop

Calibration, Performance, and Self-Improvement

V_l	Set of initial bow forces which are too low	V_h	Set of initial bow forces which are too high
V_m	Set of initial bow forces which are moderate	c'_m	Mean of timbre judgements c' of a note
V_m^-	Minimum V_m	V_m^+	Maximum V_m
a_o	Alteration of initial bow force for each note		

Preface

I have always been interested in tools to assist human creativity. In some fields of art, there are virtually no technical barriers to creativity: Free-hand drawing can be done with pen and paper; written fiction can be performed with a text editor on any computer and uploaded to any number of free blogs or websites devoted to fiction. When it comes to sheet music, LilyPond allows composers to create high-quality PDFs of their work; there is no barrier here. However, few people can look at a musical score and fully imagine how the music will sound. Sheet music is typically synthesized from MIDI with sampling synthesis, which is a poor imitation of the sound of real string instruments. The alternative is to hire live musicians to perform and record the sheet music, but this is an expensive undertaking. This work attempts to bridge this gap, at least for stringed instruments.

When reading academic papers about audio synthesis, certain papers stood out much more than others. Papers with plots, diagrams, and most importantly audio examples were much easier to understand than those lacking such media. This is particularly useful in the interdisciplinary field of music technology, where practitioners have backgrounds ranging from music, electrical engineering, psychology, or computer science. The addition of a few pictures can make a huge difference in the accessibility of a piece of research.

In order to contribute and be part of the world of good, accessible scientific publications, I spent a great deal of effort making this dissertation accessible. Whenever possible, the story is told through plots, diagrams, mathematics, and audio, rather than long textual descriptions. In addition, all source code and data is available under permissive copyright licenses, allowing anybody in the world to use, examine, or extend this research.

Acknowledgements

None of this work would have been possible without the SORSAS (Scottish Overseas Research Students Award Scheme) award from the Scottish Funding Council and a scholarship from the Faculty of Engineering (now renamed to the College of Science and Engineering) of the University of Glasgow. I am grateful for their financial support which allowed me to pursue this PhD degree.

My adviser, Dr. Nicholas Bailey, supported me with many discussions, assistance with lab equipment, and refresher lessons on physics (as my previous exposure to physics ended in first year of university in 1998). In addition, he wrote the first version of the C++ physical modelling code which eventually grew into Artifastring. That code was absolutely vital for this work; I could not have made the mental connection between the physics equations and source code without his guidance.

The physical modelling is an extension of the PhD thesis of Dr. Matthias Demoucron. I bear a considerable debt to his work: Many researchers have created physical modelling algorithms for the violin over the past ten years, but Demoucron also explained his algorithm and gave physical constants. In addition, he made numerous audio examples of his modelling available, giving me confidence that his algorithm was a good starting point.

I owe a huge debt to my parents — not only for the usual reasons that one is indebted to one's parents, but also for their specific help with this research. My mother was the primary proofreader and discussion partner for this dissertation. She was well qualified for this task, having received a B.A. in Mathematics from Oxford, a Ph.D. in Education from Simon Fraser University, and being an orchestral clarinetist and beginner violinist. My father provided practical assistance with the physics experiments and discussion of the analysis of the modal decays, drawing upon his experience as a professor of physical chemistry and his research on muon spin spectroscopy.

George Tzanetakis and Colin Percival assisted with discussions of problems in machine learning and computer programming. John Williamson explained some unclear elements of the bow friction model, while Carl Sorensen and David Kastrup aided in discussion of the puzzling effect of a single finger on the string model, leading to the use of two finger forces in the simulation. Additional proofreading assistance was given by Janek Warchol and Christopher Steinbach.

I also thank those who allowed me to borrow their instruments in order to measure their physical constants: Colin Percival (violin-I), Jennifer Leong (violin-III and violin-IV), Irene Percival (violin-V), and Wilfried Schmidt (cello-II). The other instruments were my own.

Finally, I thank all the authors of Free / Open Source software, ranging from Knuth's TeX in 1978 to complete operating systems such as Ubuntu. Of particular use to this thesis were python / scipy, marsyas, weka, LLVM / clang, valgrind, L^AT_EX/ PGF / TikZ, LilyPond, git, and blender. I offer all the code I wrote for this dissertation as a humble addition to the enormous wealth of Free / Open Source Software in the world.

Author's Declaration

I declare that my fees have been paid and that all work is mine, except where there is acknowledgement and reference to the works of others.

Chapter 1

Introduction

Western bowed string instruments (the violin, viola, and cello) produce a wonderful range of sounds. However, those sounds are difficult to reproduce with automated systems such as computers. As a result, these sounds are only created by highly-trained humans with thousands of hours of experience. Anybody wishing to create violin sound must either hire an expert or become an expert themselves.

There are two main challenges faced by computer synthesis of violin family sounds, both stemming from the instruments creating sound based on continuously varying parameters (e.g., the bow scraping against the string, the air flow into a clarinet) rather than a single event (e.g., a guitar pluck, a drum hit). First, the behaviour of mechanical systems with continuous excitations is harder to describe mathematically than systems with a single excitation. Second, due to the amount of control values required to describe the excitation, some sort of algorithm must be used to provide these values.

This dissertation presents significant advances to both problems by creating virtual musicians to control virtual stringed instruments. My use of the term “virtual musician” is illustrated in Figure 1.1 and formalized as:

Given a machine-readable representation of sheet music, the computer autonomously produces audio and video that sounds as if it was performed by a human.

I am using the term “autonomous” to mean that human input may be used to train the computer program but will not be used directly in the process of generating audio from sheet music. By analogy, consider a human musician: Although musicians benefit a great deal from teacher input, concerts are given without teachers shouting commands in the middle of the performance.

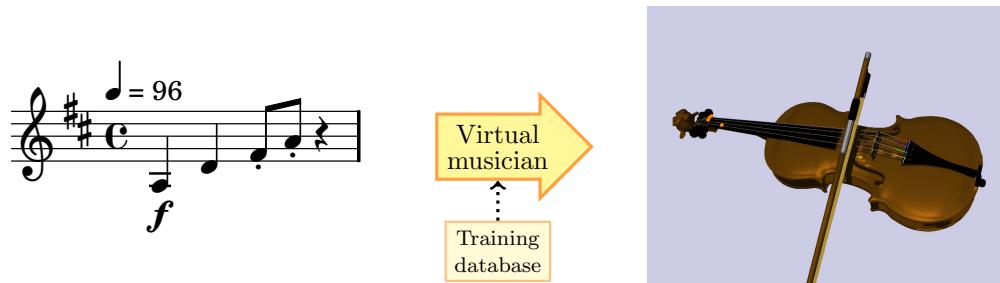


Figure 1.1: Sheet music to video with no human interaction.

Video 1.1: Black box performance

<http://percival-music.ca/dissertation/v.1.1.black-box.mpeg>

1.1 Motivation and applications

My interest in this problem arose from four distinct areas: Musical robotics; Vocaloid and UTAU computer singing software; accessibility and assistive technologies; and finally my experiences as a music student with composers, performers, and musicologists. I was also highly influenced by the Free / open source software, copyleft art, and the open access movements. I shall examine each area, discuss on how they relate to my work and list potential applications in their fields, then re-examine the overall problem definition.

1.1.1 Musical robotics

The first topic in music technology which truly ensnared my imagination was musical robotics: Autonomous machines which play unmodified musical instruments. Some musical robots were created to enable serious scientific investigations in acoustics or musical performance, but many were created for the obvious goal of “showing off” current technology.

I will begin by discussing a few robots which do not include any audio-based feedback control or “intelligence” — they simply reproduce a series of pre-determined physical actions. This will be followed by a survey of robots using audio feedback control — robots which adjust their physical actions in response to the sound.

Musical robots without feedback control

An early violin-playing machine was created by the Nobel-prize-winning physicist C. V. Raman. In addition to his research in optics, Raman was fascinated by sound, publishing works on theoretical and experimental acoustics from 1909 to 1936. In particular, he constructed a machine from discarded bicycle parts and old laboratory materials in order to test his theoretical predictions. Unlike normal violin playing, this machine held the bow immobile and moved the violin along a track (Raman 1920).

A much more recent example is a robotic clarinet player (Almeida, Lemare, Sheahan, Judge, Auvray, Dang, John, Geoffroy, Katupitiya, Santus, Skougarevsky, Smith & Wolfe 2010), a project which began as an entry to the ARTEMIS Orchestra Competition of 2008. This competition is run by an industry association for embedded electronics and is seen as a way to demonstrate the flexibility of modern electronics. After winning the competition, the researchers used the robot to investigate the behaviour of a clarinet given strictly-controlled air pressure and lip force.

Students are naturally interested in musical robots; in 2006 a team of fourth-year mechanical engineering students created a “RoboFiddler”, a robotic violin player (Chia, Hong, Lee & Lim 2006). The playing ability of RoboFiddler is comparable to a student with two or three weeks of experience, but due to the difficulty of the task the students deservedly won their university’s “Best Mechatronics Project 2006” award. Furthermore, they took the unusual step of putting their final report online, containing detailed designs for hardware and software used.

The Toyota corporation is actively developing musical robots as promotions for their “partner robot” initiative to sell household robots. They demonstrated a trumpet-playing robot in 2005 and a violin-playing robot two years later (Kusuda 2008) which sounds like a student with three or four years of experience. These projects by Toyota are impressive feats of mechanical engineering: Unlike most musical robotics projects which fix the instrument in a frame to reduce complexity, they used humanoid robots which hold the instruments in the normal fashion.

Musical robots with feedback control

The theremin is an electronic instrument which operates with no physical contact. The theremin has two metal antennae; one controls pitch while the other controls volume. The distance between the performer's hands and each antenna forms a capacitor, constituting part of a tuned circuit whose electrical oscillation is amplified to produce the audio output. As such, the surrounding electromagnetic field can alter the pitch produced by each distance. (Wu, Kuvichkul, Cheung & Demiris 2010) used a robot with 2 degrees of freedom (DOF) to play a theremin, with dynamic programming to select the minimum energy required for pitches, and feed-forward control to adjust for environmental changes.

The Waseda Saxophonist Robot No. 2 (WAS-2) (Solis, Takanishi & Hashimoto 2010) is a much more complicated robot, comprising 3 DOF for the artificial lips, 16 for the hand and fingers, 1 DOF for the tongue, and 2 DOF for the lungs. A few proportional-integral-derivative (PID) controllers are used for low-level mechanical control, while the higher-level control is implemented with an artificial neural network (ANN) with feed-forward error learning and a dead-time factor to compensate for delay in the air pumps.

The same team of researchers also developed a flute-playing robot. The Waseda Flutist Robot No. 4 Refined IV (WF-4RIV) (Solis, Taniguchi, Ninomiya, Petersen, Yamamoto & Takanishi 2009) has even more complicated mechanics, with a total of 41 DOF compared to 22 DOF for the WAS-2. Their control system is based on three separate ANNs. The first specifies the duration of each note, the second specifies the vibrato duration, while the third specifies the vibrato frequency. Audio analysis of pitch, the strength of even and odd harmonics, and overall sound intensity level is taken into account in order to determine an overall “quality” for each note; if a note’s quality is too low, the robot searches for improved parameters to use for the next performance.

Relation to my work

Musical robots are fascinating, yet at the moment their manufacture requires expensive and resource-intensive work. In a few decades, robots may be sufficiently inexpensive that every household will have a humanoid robot capable of reproducing the mechanical actions required to play musical instruments, but for now they only exist in robotic research laboratories. I can side-step the problem of access to actual robots by using physical models (simulations) of stringed instruments. If the physical models are sufficiently accurate and my control algorithms are sufficiently general, my work on virtual musicians can be applied to robots playing real stringed instruments.

1.1.2 Computer singing synthesis

The human voice is the hardest “instrument” to synthesize — not only are there many control parameters (e.g., lips, tongue, throat, vocal chords, air pressure leaving lungs) which affect the pitch and timbre, but these change very rapidly to form different syllables. Fortunately, there has been a great deal of research and commercial interest in computer synthesis of speech and singing.

It has been fascinating to watch progress in this area over the past decade. Ten years ago, computer singing synthesis in research papers and commercial products were of rather questionable musical value. Their quality has improved dramatically in recent years, and there are now thousands of high-quality songs and videos created with computer singing synthesis.

Vocaloid and UTAU

The most famous commercial singing synthesis is Vocaloid (Kenmochi & Ohshita 2007), popularized by the “virtual pop idol” Hatsune Miku. The history of vocaloid’s popularity is given in (Kenmochi 2010). Vocaloid uses Spectral Modelling Synthesis (Serra 1989), in which short portions of recorded audio are “stitched together” using various digital signal processing techniques; more details are given in Section 1.2.3. This program provides realistic singing for certain styles of music (notably Japanese pop music). In addition to the commercial Vocaloid synthesizer, a shareware program called UTAU (the Japanese word for “song”) allows users to record and share their own singing voice.

These two programs, as well as the commercial and non-commercial recordings of voices, have allowed users to create a huge amount of music and videos. It should be noted that Vocaloid and UTAU themselves do not sing in an “expressive” manner; to improve the audio, a human must carefully alter note parameters (e.g., pitch, vibrato, onset, duration). Such projects often involve collaboration over the internet: One person may write some lyrics, another will set the lyrics to music, a third person will make micro-adjustments to the singing parameters, a fourth person will create the guitar and drum tracks, a fifth person will draw images and create 3-D computer models, and a sixth person may create a music video combining the audio with computer-animated dancing. Since these collaborations generally take place on a specific website¹, researchers have data-mined meta-data from this resource to find interesting patterns in the collaborations (Hamasaki, Takeda, Hope & Nishimura 2009).

Vocaloid has been used directly in several research projects. Vocalistener2 (Nakano & Goto 2011) automatically extracts singing parameters from recorded audio; this allows users to reproduce a song “the way they sang it” using completely different voices. Vocawatcher (Kajita, Nakano, Goto, Matsumasa, Nakaoka & Yokoi 2011) takes this one step further: In addition to reproducing the sung vowels and musical expression, this software tracks head and facial movements (e.g., blinking or closing eyelids, position of upper and lower lips, neck angle) and reproduces them using the human robot HRP-4C (a robot with the appearance of a young Japanese woman). In this case, the robot’s physical actions are purely cosmetic: Audio is produced with the Vocaloid software, not with physical air pumps and mechanical vocal chords.

Relation to my work

Computer singing synthesis is one of most successful fields of music technology in the past decade. Even ignoring the number of academic citations and commercial fees from licensing the patents (two traditional measures of research success), this technology has allowed a diverse range of art to flourish. In particular, the UTAU software allows people to compose and produce songs without being limited to their own voice or facing the financial burden of purchasing commercial software. Combined with guitar, drum, and piano synthesis (already noted to be much easier to synthesize than voice or violin), this allows them to create music in a wide range of popular styles. There is a large community of Vocaloid enthusiasts online², sharing music and videos they created. There is a clear appetite for creating and enjoying music with computer music synthesis.

¹Nico nico douga, a video-sharing site used primarily in Japan. <http://www.nicovideo.jp>

²Anecdotal (non-peer-reviewed) evidence from computer analysis of nico nico douga meta-data claims that at the beginning of 2012, there were 27 videos with more than 250k views, 78 videos with between 100k and 250k views, and an overall power-law distribution. Popular videos are often re-posted on other video-sharing websites, but those views are not reflect in these figures. <http://www.vocaloidism.com/2012/01/04/the-harsh-realities-of-vocaloid-on-nico/>

1.1.3 Accessibility and assistive technologies

Many able-bodied adults take physical actions for granted — we give no thought to walking up stairs, reading text on a computer screen, or controlling a computer mouse. However, some people have difficulty with such basic tasks. Fortunately, we can use technology to assist their lives. I will examine a few ways in which technology can assist interaction with music.

Assistive music technology

In order to listen to music, many people select a song on their computer or music player. Clicking on a song in a media-playing program does not require a great deal of physical force, but some people lack the fine motor skills to control a digital pointing device. Even worse, if a song has little or no meta-data, selecting the desired song can require multiple clicks. Work such as (Tzanetakis, Benning, Ness, Minifie & Livingston 2009) uses audio feature extraction and machine learning to categorize a music collection, then presents the results in a self-organizing map to make music browsing easier. Music creation requires more interaction than merely selecting the type of song to listen to. One project tackling this problem is the Eyeharp (Vamvakousis 2011), which used a very cheap device (less than \$100) to perform eye tracking for patients with limited mobility. Their eye movements control specialized software to create audio, enabling even people paralyzed from the neck down to compose and perform music. The notion of accessibility is taken one step further in (Miranda, Magee, Wilson, Eaton & Palaniappan 2011), which used a brain-computer music interfacing system to create music according to the electroencephalogram (EEG) activity of a patient. EEG hardware is quite affordable; their entire setup (including laptop) was less than \$3500.

This type of research has benefits beyond the small portion of the population with serious disabilities. As humans age, our physical capabilities decrease. We lose lung capacity, arm strength, and finger agility. We can prolong our music-making activities with regular exercise or by switching instruments (i.e. saxophone to recorder) and playing simpler music (i.e. concertos to simple folk tunes), but after playing an instrument for 50 or 60 years, playing one-octave melodies on a recorder may seem sub-optimal. Musical creativity becomes hindered by physical constraints.

Some people may object that such systems would impinge upon the “purity” of music — that music should only be performed by humans using the instruments for which the music was composed. However, I reject this argument; it is tantamount to saying that only physically fit people “deserve” to produce violin sounds. I argue that music is something that everybody should be able to enjoy, and that allowing more people to create music should not be viewed as a bad thing.

Relation to my work

A virtual musician combined with input devices would allow us to “offload” many physical challenges. For example, a bed-ridden patient could control aspects of the music with a computer mouse, such as mapping the two dimensions to speed and overall loudness. More sophisticated input devices could allow the user to specify bow force, velocity, and speed of left-hand vibrato. Users with less physical agility could practice music at half speed but have their movements synthesized and performed at normal speed. Alternately, the virtual musician could be trained or given overall “musical” direction (even with only gaze tracking), then it would automatically fill in all finger and bow movements in order to fulfil the high-level musical desires.

1.1.4 Audio synthesis for composers, performers, and musicologists

While learning cello as a child and teenager, it never seemed to me that there was any use for computers in music. My life revolved around string quartets and orchestras; even if a computer could produce a decent violin sound, it would not be able to follow the natural shifts in tempo that occurs in classical music. If I wanted to listen to a recording of music to help me learn it, I could buy a CD of some of the world's best musicians performing it. This view changed drastically when I began studying music composition at university.

Aiding music composition

Students wishing to learn how to compose music for stringed instruments face a difficult task: Computer synthesis of these instruments is poor, but skilled musicians are quite rare and thus expensive to hire. Having a “reading session” where real musicians perform new compositions is a major event in a young composer’s life; this may occur as often as once a month, but it is not uncommon to have only two or three sessions each year. It is commonly accepted that composers must learn to “hear the music in their mind”, but this is a significant challenge for most students. I played many new compositions by my classmates, and they were often surprised at how their music sounded. It would be useful if music composition students could hear decent audio performances — not perfect performances, but *credible* performances — of their works more often.

This could also be useful for “casual” composers: People who are not expecting (or even desiring!) a career in music composition, but who simply enjoy creating music. Having seen the impressive music and artwork that Vocaloid has fostered, there is strong evidence that music synthesis does not need to be perfect to enable good art.

Aiding performers

Such a tool could be useful for performers. As previously noted, musicians often use audio recordings to help them to learn a piece of music. For classical music, many such recordings already exist. But when learning a piece of new music, no recordings exist. A computer performance will not be as “musical” as a recording from a professional musician, but it could still be a great aid for memorizing newly-composed music. Such a performance could be desirable even when learning previously-recorded pieces of music — a musician might prefer to learn from a “neutral” recording which does not contain any individual interpretations which other performers added to their playing.

Computer simulations could also aid violin students who are beginning to plan a musical interpretation. Very young students simply perform music with no attempt at personal musical expression (by either following their teachers’ guidance exactly, or by simply “playing the notes” with no attempt at expression), but more experienced students are expected to add musical expression themselves. This often involves choosing different bowings, playing on different strings, and trying different fingerings. It can be quite challenging to play a difficult piece of music while also trying novel bowings or fingerings; a computer simulation could allow students to hear the effects of various bowings or fingerings without the burden of learning how to perform the experimental bowings themselves.

The ability to simulate different bowings and fingerings could also aid more experienced compositions students. In addition to allowing compositions students to experiment with various options before hiring expensive musicians, an synthesized recording could improve communication between

the composer and musician. Many parts of classical musician pedagogy are still “oral tradition” — although there are written accounts of musical style, almost all musicians learn from hearing teachers and more experienced musicians talking and demonstrating the style. When music performance students play new pieces of music from music composition students, there is often a communication gap between the performers expecting audio examples and composers showing the sheet music. If the composers could create an audio recording that demonstrates the style, such confusion would be greatly reduced.

Music libraries

Music libraries could benefit from virtual musicians able to perform sheet music. Although many users of such libraries will be able to “visualize” how a piece of music will sound by reading the score, this ability diminishes as the number of parts increase. Some pianists are trained to perform a “condensed” version of a full orchestral score with no preparation (“sight-reading”), but this skill is rare and requires a piano and extra effort. It would be very useful if the web search of a music library provided recordings of all music in the collection³. Hiring musicians to perform all the works would cost a great deal of money and could pose copyright problems (i.e. the performers’ copyright may restrict access to those music recordings). Both problems could be avoided with virtual musicians.

Virtual musicians could be a great aid to musicology as well. In addition to benefiting from a “quick and dirty” performance of scores which exist only in manuscript form, if the virtual musicians were sufficiently advanced, the musicologist could select different performance characteristics. What would a particular 1730s Minuet sound like if it was performed on instruments with gut strings with a string quartet which began every measure with a down bow? What if the cello part was performed on a viola de gamba instead? What if the same work was performed by a dozen musicians, with a conductor beating time with a staff on the floor? If a musicologist wanted to experiment with these parameters at the moment, they would need to spend a great deal of money for each recording session, but a virtual musician could allow them to simulate different parameters at no cost.

1.1.5 Free / open source software, copyleft art, and open access

We live in an age of incredible virtual wealth due to the free / open-source software movement (Stallman 2010). Thousands of programmers have written software released under copyright licenses which permit copying and modification (sometimes known as “copyleft” licenses). With a cheap computer, an internet connection, and the willingness to learn, anybody can legally download software enabling them to write books, design software, compose sheet music, produce videos, and do thousands of other artistic tasks. In some areas, free / open-source software is generally seen as superior or comparable to commercial programs (e.g., web browsers, software compilers); in other areas, commercial software still dominates (e.g., architecture drawing programs, games). However, if one is highly constrained by financial resources but does not wish to infringe on copyright, there is almost always a way to perform any desired digital activity using free / open-source software. I am very encouraged by the array of tools available: In an ideal world, the only thing restricting human creativity should be human creativity itself.

³For example, the International Music Score Library Project (Project Petrucci LLC 2012) currently has over 55,000 musical works freely available, yet only slightly over 17,000 recordings.

The related “copyleft art” movement holds the same ideals: By freely sharing artwork (be it literature, drawings, music, or movies), we can increase the “cultural wealth” of the world much more than trying to charge money for every piece of art. Attempting to monetize everything covered by copyright law results in additional burdens for artists who need to check that their new works do not infringe on anybody’s copyright. A typical example is a film director needing lawyers to check every object in a movie scene — every object covered by copyright (e.g., statues in the background, the design of a particular type of chair, adverts on the side of a milk carton) needs to be licensed. This is a boon to lawyers seeking work, but adds a huge burden to artists (Lessig 2001). The copyleft movement seeks to reduce or eliminate this burden by providing artwork which allows redistribution under permissive terms, often with one of the Creative Commons licenses (Fitzgerald, Coates & Lewis 2007). This is not the default position in every country; for example, in Japan there is a general acceptance that modest use of other people’s art is acceptable provided that credit is given. In fact, it is even accepted practice that fan-created art (called *dōjinshi*) using characters from commercial works can make a modest profit (Tushnet 2007).

My research has benefited immensely from free / open-source software and the copyleft art movements. In terms of software, I have used compilers, profiling tools, signal processing libraries, audio tools, and video rendering. In terms of art, I have used works from the vast collections of copyleft sheet music, allowing me to test my virtual musicians on a wide range of music. There is nothing special about the story so far — I could probably replicate all the above software and sheet music by purchasing commercial software and sheet music at a cost of “merely” \$5,000 - \$10,000. However, commercial software very rarely gives the user the ability to modify the software itself. In addition to being available at low cost, my ability to modify free / open-source software has been invaluable. For example, I added some additional digital signal processing features and machine-learning tools to the Marsyas library (Tzanetakis 2007). Without the ability to modify the source code, I would have needed to either completely rewrite the software myself (taking years of effort), or else make a feature request to a commercial vendor (requiring years of waiting, if the vendor implemented my requests at all). That would have rendered my project impossible to complete within any reasonable time frame.

Related ideas are changing academic publishing: The Open Access movement is encouraging researchers to make their papers available to anybody free of charge. This takes two forms: “Gold Open Access” means that the publishers themselves allow free access, while “Green Open Access” means that the researcher has made a copy of their article (or possibly an unofficial “pre-print”) freely available (Laakso, Welling, Bukvova, Nyman, Björk & Hedlund 2011). There are two main arguments in favour of open access. The first is a moral argument: The free spread of knowledge is a public good; science flourishes when there is free debate. Universities in Western countries may complain about the journal fees, but ultimately they can be paid — but there is little hope of universities in developing nations paying for the same access. This is particularly relevant when we consider medical research (Chan, Kirsop & Arunachalam 2011). In addition, most research is funded by tax-payers, either directly through grants or indirectly through salary. It seems unfair to deny tax-payers the ability to read papers from their funded research. The second is a practical argument: Open access increases the impact factor and increases citation count. There has been some debate about whether this advantage is causal or simply due to self-selection (some academics may choose to give open access to only their best works). However, most studies suggest that the advantage is indeed real (Gargouri, Hajjem, Larivière, Gingras, Carr, Brody & Harnad 2010).

1.1.6 Research constraints to generalize applicability

Having discussed potential applications, I can clarify my project. I am creating a virtual string quartet in order to perform music for inexperienced or casual music composers, musicologists, or musicians who are no longer physically able to perform their normal instruments. Although the musical instruments will be synthesized using computer software, the “virtual musicians” should be sufficiently general such that they could be used to control musical robots performing unmodified musical instruments.

In the spirit of open scientific progress, I resolved to make this work as accessible as possible. This led to the following (voluntary) constraints:

Free software: The entire system must be available at no cost, and furthermore, everybody must be legally allowed to modify it (“free” as in “freedom”).

Easily extended: The system should be clearly designed with good documentation, such that a moderately skilled programmer can add new features. Any data required for the system should be as easy to gather, without requiring any expensive equipment or special acoustic environment.

Human-like pedagogy: The system should be trained in a manner similar to training a human student, since most musicians are experienced at teaching humans but are not programmers.

1.2 Literature review

This section covers publications relevant to methods and techniques I use in this dissertation. I shall briefly summarize research on violin acoustics and violinist actions, computer synthesis techniques, musical applications of feedback control, music information retrieval, and finally music education and expressive performance.

1.2.1 Violin physics and mechanical control

The violin is a mechanical system: The violinist’s fingers and bow interact with the strings, which eventually causes certain vibrations to reach our ears. How do vibrations from the strings reach our ears? How do the finger and bow interact with each string? What physical actions must musicians perform in order to achieve the desired sound?

Historical era (pre-1985)

Bowed stringed instruments have interested physicists for over two thousand years. The ancient Greeks studied acoustics, but the first revolutionary step towards our current understanding of these musical instruments came from d’Alembert’s solution to the wave equation in 1747. Over a century later, Helmholtz discovered that when the violin produces a good tone, the string is moving in a “V-shape” in which the corner of the “V” (now known as the “Helmholtz corner”) travels from end to end of the string. Ideally, the bow should be sticking to the string most of the time, only slipping when the “V” is between the bow and the bridge. This process (now known as “Helmholtz motion”) repeats once per cycle of the resulting sound. For example, when the violin open A string (440 Hz) is played with a good tone, the string undergoes a stick-slip-stick transition 440 times each second (Helmholtz 1895).

The next major step in our understanding of vibrating strings came from Raman, who investigated the bow-string interaction. After making theoretical predictions (assuming a perfectly flexible string,

excited at a single point), Raman built a mechanical device to play the violin with the bow at a specific distance from the bridge and specified speeds (Raman 1920). Using his device, he investigated various combinations of bowing position, pressure, and bow velocities.

Real-life strings are not perfectly elastic; string stiffness results in a “spreading” of the frequencies of upper partials. A perfectly elastic string will have upper partials of frequencies $f_n = n f_0$, where n is the partial number, f_n is the frequency of that partial, and f_0 is the frequency as predicted from string length and tension alone. However, (Fletcher, Blackham & Stratton 1962) found that given the string inharmonicity constant B ,

$$f_n = n f_0 \sqrt{1 + B n^2} \quad (1.1)$$

The violin body has a long history of research as well. In 1787, Chaldni observed patterns of nodes and anti-nodes by sprinkling sand on plates from violin and guitar bodies and bowing the side. Almost two centuries later, the plates were excited with signal generators attached to loudspeakers, and then still later, hologram interferometry was used (Hutchins 1981).

Other physicists throughout the 20th century studied the interaction of string vibrations with string terminations, fingers on the string, and of course friction with the bow. I mark a rough boundary in the mid-1980s between the “historical era” of acoustics research and the “current era”. There are two reasons for this boundary: First, the increasing power of computers simulations allowed a vast shift in the type of research that was possible — scientists could simulate activities such as bowing a string or blowing into a clarinet, then listen to the resulting (simulated) sound. Second, there were two seminal publications which framed the next decades of research.

One of these publications was a book which covered the state of violin physics (Cremer 1984). This provided a stable foundation for new researchers to become familiar with the previous work. The other seminal publication was a journal article (McIntyre, Schumacher & Woodhouse 1983), which introduced a general model for vibrations in musical instruments now known as the MSW model. This model allows for various types of non-linearity, such as pitch flattening (where the bowed string produces frequencies slightly lower than would be predicted by a simpler theoretical model), subharmonics (where a bowed string produces energy at frequencies which are half of the expected lowest frequency), and “wolf notes” (where the string alternates between Helmholtz motion and having two “slips” per cycle, resulting in a highly unsteady tone; this occurs due to an unfortunate coupling between string modes and instrument body modes). A great deal of later research, especially in the area of computer simulations, relied on the MSW model.

Current era (1985 onwards)

A general introduction to the physics of stringed instruments is given in (Rossing 2010); this book is also a good source of physical constants for strings and instrument bodies. An excellent review paper for academics is (Woodhouse & Galluzzo 2004), covering both the history of bowed string physics research and current research questions. Another good source of physical constants, this time aimed at instrument makers, is (Jansson 2002); in addition to giving an overview of acoustics, this work discusses the effects of different materials on string inharmonicity and tension. A standard reference for physical properties of violin strings is (Pickering 1985), although Pickering does not examine the inner construction of wound heterogeneous strings. (Firth 1985) examined disassembled wound strings

with a scanning electron microscope to make detailed measurements of the core diameter and the layers of fibres and solid wrapping.

The effects of string inharmonicity were studied in (Jarvelainen, Välimäki & Karjalainen 2001) by performing listening tests with synthesized piano tones. They found that string inharmonicity was important for lower-frequency notes, but that for high-frequency notes the inharmonicity found in real instruments was very close to the threshold of perceptibility and thus it may be possible to omit them during audio synthesis with no loss of perceptual quality.

In addition to transverse vibrations, strings vibrate in other directions. Torsional vibrations in cello D strings were investigated in (Woodhouse & Loach 1999), finding that torsional vibrations appeared to occur at harmonically-spaced frequencies. However, they were only able to measure the first 7–10 modes of torsional vibration on three tested strings, so any distinction between inharmonic peaks and harmonic peaks would not necessarily be detectable in this range. Interestingly, the Q factors of torsional decay were almost constant, decreasing only slightly throughout the measured modes.

The interaction between the bow and string is quite complicated. As the bow scrapes against the string, the friction generates heat which partially melts the rosin on the bow; this reduces the friction coefficient (Smith & Woodhouse 2000). This heat causes hysteresis in the tribology of rosin. Although great progress has been made in our understanding of the bow-string interaction, some mysteries remain. One such mystery is the noise component: instrumental sounds contain varying degrees of white noise. A complete physical description of the instrument’s mechanics would explain this noise, but until our understanding of the mechanics reaches this point, sound synthesis can be improved by adding noise to the relevant part of the modelling. In particular, (Chafe 1990) found that multiplying the force of friction by a scaled uniform noise term $u(t)$ produced bursts of noise when the bow direction changed, which matched experimental measurements. Simulations showed that including uniform noise in this fashion aids the production of sub-harmonics.

Close examination of the spectrum of plucked and stuck strings reveals a curious phenomenon of “phantom partials” (Conklin 1999) or “split peaks” (Penttilä, Pakarinen, Välimäki, Laurson, Li & Leman 2006) wherein there appears to be energy at both the frequency predicted by string inharmonicity and either the ideal frequency or a frequency predicted by a modified equation for string inharmonicity.

The effect of vibrating modes in violin bodies have been studied (Fritz, Cross, Moore & Woodhouse 2007). A violin was played normally while they measured the vibrations in the bridge (thereby avoiding the violin body). That signal was transformed with different filters corresponding to admittance curves (impulse response measured at the bridge) of different violins, with various modifications to the amplitudes and frequencies of body modes. Musicians noticed modifications to the modes as little as 3 dB for amplitude changes and 1.5% for frequency changes.

Accurate measurement of the body impulses were improved by plucking the strings close to the bridge with a thin loop of wire rather than hitting the side of the bridge with an impact hammer (Türckheim, Smit, Hahne & Mores 2010). The effect of the instrument’s bridge (rather than the instrument body) was studied in (Woodhouse 2005).

The violin strings and body are not the only factors involved in the production of sound. The violin bow is a resonating structure whose behaviour can greatly alter the musician’s ability to produce a good tone (Guettler & Askenfelt 1995). Even the choice of rosin placed on a bow can alter the resulting sound and ease of controlling the instrument (Guettler 2011).

Mechanically controlling a violin

The vast majority of violin playing is done with a bow, so violin research has focused mainly on the bow-string interaction. There is a substantial body of research onto plucked guitar strings, which is still applicable to the violin. As noted earlier, Helmholtz found that “good tone” occurs when the string undergoes a stick-slip-stick cycle once per cycle in the final sound — i.e. when playing the open violin A string (pitch 440 Hz), good tone is achieved when the string slips (relative to the bow) 440 times per second. The natural follow-up question is “under what conditions is this achieved?”.

The first major step in this direction was (Schelleng 1973), which found theoretical limits for the minimum F_{\min} and maximum F_{\max} bow forces which could establish Helmholtz motion in sustained bowing given the early 1970s understanding of violin string mechanics. If the bow is moving at velocity v_b at a relative bow-bridge distance β ,

$$F_{\min} = \frac{Z_0^2 v_b}{2R(\mu_s - \mu_d)\beta^2} \quad F_{\max} = \frac{2Z_0 v_b}{(\mu_s - \mu_d)\beta} \quad (1.2)$$

where the physical constants are the characteristic transverse impedance of the string Z_0 , static coefficient of friction μ_s , dynamic coefficient of friction μ_d , and Raman bridge resistance R . The Schelleng bow forces are almost always referred to, and visualized as, “the Schelleng diagram” showing these lines in a plot of β vs. $\log F$.

Although Schelleng presented plausible theoretical limits on bowing force, these were not measured experimentally until (Schoonderwaldt 2009). Schoonderwaldt constructed a bowing machine to test the steady-state bowing response of a violin in order to generate “empirical Schelleng diagrams”. He found that the theoretical predictions for F_{\max} provided a good match to empirical evidence, but that there were major deviations between the theoretical F_{\min} and empirical evidence. In particular, the bow velocity was not significant in the tested range (0.05–0.20 m/s), the empirical F_{\min} was almost an order of magnitude higher than the theoretical prediction, and the amount of damping in the string was much more important than predicted. This discrepancy lies in Raman’s assumption (adopted by Schelleng) that the string terminations are purely resistive.

Note attacks were studied in (Guettler 2002), who made theoretical predictions and performed computer simulations to determine a “perfect” attack which establishes Helmholtz motion immediately (i.e. no extraneous slips). These are often visualized as “Guettler diagrams”, showing triangular areas of perfect attacks in a plot of bow acceleration vs. bow force for a fixed β . Helmholtz motion is the ideal, but real musicians do not always achieve this motion, particularly in the initial note attacks. Notes which established Helmholtz motion within 50 ms are perceived as being acceptable by advanced string students. It is estimated that between 20%–50% of notes from professional violinists have a perfect attack, although 80%–90% achieve Helmholtz motion within 50 ms.

Wolf notes have been studied by using modal simulations of strings and the instrument body to find regions in the space of bowing parameters (i.e. bow-bridge distance, bow speed, and bow force) which produce good or bad tone (Inácio, Antunes & Wright 2008). Such studies may be able to help musicians avoid wolf notes, but the main application is to help instrument makers construct instruments whose modes of vibration avoid the unfortunate coupling leading to such problems.

In order to track the physical actions which musicians actually perform — instead of investigating the limits of violin playability — accurate, small, and lightweight sensors are needed. This has been an active area of research in recent years. (Young 2007) installed a measurement system (weight 23 g)

inside a carbon fibre violin bow, used its data to classify bow-strokes with machine learning (accuracy over 90%), and stored that information in a web-accessible database. (Rasamimanana 2008) created a measurement device (weight 17 g) which clips onto an unmodified bow and performed similar machine-learning classification of bow-strokes. (Demoucron 2008) designed a lightweight (3.8 g) sensor for bow force measurement which can be easily attached or detached from any bow, and generalized bowing parameter contours for bow-strokes into mathematical formulae. (Schoonderwaldt 2009) added a motion capture system to Demoucron’s sensor by adding reflective markers to the bow (attached with a special adhesive which left no marks on the instrument varnish), providing a highly accurate yet lightweight (10 g) system for detailed measurements. (Maestre 2009) and (Pérez 2009) used a commercial tracking system based on electromagnetic field sensing with numerous sensors which provide data about sensor translation and rotation (adding 12 g to the weight of the bow).

Research in acoustics and examinations of violinists’ actions are having an effect in performance and education. Anomalous Low Frequencies (sometimes incorrectly called “subharmonics”), in which the string vibrates at frequencies below the fundamental frequency, have been used by a professional violinist (Kimura 1999). Special exercises for violin students, informed by knowledge of Helmholtz motion and Schelleng’s predictions, were created in (Collins 2009).

1.2.2 Violin physical modelling

This section discusses using equations from acoustics research to simulate the behaviour (and thus the sound) of musical instruments; this idea was mentioned briefly in the previous section. Two standard reference books on physical modelling and audio signal processing are (Smith 2010) and (Cook 2002).

Digital waveguide synthesis

Most early physical simulations relied on calculating numerical solutions of the wave equation, which is computationally expensive. In contrast, digital waveguide synthesis relies on d’Alembert’s solution to the wave equation: The wave is modelled as two separate waves, one travelling “right” and the other travelling “left”. Given the displacement $y(x, t)$ at position x at time t , with c being the wave’s speed,

$$y(x, t) = y_{\text{right}}(x - ct) + y_{\text{left}}(x + ct) \quad (1.3)$$

Assuming that the wave’s behaviour in the system is linear time-invariant, each travelling wave can be modelled with a delay line with any losses along the string or in the reflections at the ends of the string combined into a single filter. To find the displacement at any particular point, the two waves are summed together. To bow the string, the delay lines are “split” at the point of contact between the bow and string. Two delay lines represent the travelling waves between the bow and the bridge, while another two represent the travelling waves between the bow and the nut.

This provides a solution which is remarkably simple, both in terms of programming effort and computation requirements. Even in the early 1990s, multiple instruments could be simulated on a single inexpensive DSP chip (Smith 1992). Digital waveguide synthesis formed the basis of the popular Synthesis Toolkit in C++ (Scavone & Cook 2005), which provided researchers and musicians with a common baseline of physical modelling.

The changing friction characteristics from melting rosin were modelled, along with the finite width

of the bow and a waveguide mesh for the instrument body, with waveguide synthesis in (Serafin 2004). Different friction equations were investigated by generating Schelleng diagrams to visualize the “playability” (i.e. the size of regions where Helmholtz motion was possible). (Sterling 2010) measured the bridge input admittance and the radiation transfer function of a violin and used these to improve the output the digital waveguide synthesis.

Modal synthesis

Another popular, although more computationally expensive, method of simulating a vibrating string is to use modal synthesis. Harmonic sounds can be represented as a superposition of vibrating modes. There are theoretically infinitely many vibrating modes, but the effect of higher-order modes decreases dramatically. In sound synthesis, the number of modes is typically limited to between 10 and 200.

As will be discussed in Section 1.3.2, modal synthesis is the type of physical modelling used for my dissertation. In particular, I used the algorithm presented in (Demoucron & Rasamimanana 2009, Demoucron 2008). Demoucron used modal synthesis as a basis to re-create different types of bow-strokes: He first measured the physical actions of real violinists, then devised mathematical equations to express the progression of bow force, bow-bridge distance, and bow velocity over time, then synthesized new bow-strokes using those functions. Demoucron made no claim that his synthesis was unique nor cutting-edge; his focus was on controlling the system. The algorithm is largely based on (Adrien 1991) and does not include various factors such as torsional waves, the width of the bow, vibrations in the instrument body, or hysteresis in the bow-string friction.

An important factor in modal synthesis is the decay rates of individual modes. A great deal of research on this subject comes from the study of room and concert hall acoustics, in which the reverberation time and modal decay rates are crucial factors to be considered by architects. An excellent review of previous methods of detecting modal decay, and a new technique based on nonlinear optimization, is presented in (Karjalainen, Ansalo, Mäkivirta, Peltonen & Välimäki 2002). Some strings exhibit “two-stage” decay, in which a steep initial decay rate is followed by a shallower long decay; these were analyzed by finding best-fit lines to measured data and synthesizing the results in (Lee, Smith & Välimäki 2010).

The modal behaviour of guitar strings was measured and synthesized to test acoustic theories in (Woodhouse 2004). This work was later extended (Woodhouse, Manuel, Smith, Wheble & Fritz 2012) to determine the just noticeable difference (JDN) in frequency and decay rates of the modes, finding that the best listeners could detect a 1% shift in the frequency of body modes, a 20% shift in the damping of body modes, and a factor of 3 for the damping of string modes.

1.2.3 Alternate violin synthesis methods

Although the sound synthesis in this dissertation is based on modal physical modelling, it is worth briefly reviewing the two main alternatives which may be suitable for violin music. These are “data-driven” synthesis methods: Instead of using a number of “rules” (i.e. mathematical equations from physics) to generate the sound, the bulk of the synthesis is performed by using recorded audio data, optionally transformed in some way.

Sampling synthesis

Sampling synthesis consists of concatenating pre-recorded snippets of audio known as “samples”⁴. These snippets could be as short as one cycle of the desired frequency, but as hard drives and memory limits increased in the 1990s, longer “samples” were used. This is the simplest form of audio synthesis, and is very widespread in both commercial and non-commercial hardware and software (Cook 2002). Sampling synthesis now dominates the commercial music industry, with multi-gigabyte sample libraries providing a reasonable facsimile of Western orchestral instruments and some non-Western instruments (Garritan Libraries 2012, Vienna Symphonic Library GmbH 2012).

The strength and weakness of sampling synthesis is that the audio is played back exactly as it was recorded. If this matches the desired sound — for example, pressing a particular piano key — then sampling synthesis captures all non-linearities of the sound. However, if the desired sound is slightly different — for example, pressing the same piano key but with the sustaining pedal pressed, or pressing a different piano key — then sampling synthesis faces enormous difficulty to alter the previously-recorded audio. This is particularly apparent when considering instruments with continuous excitation (e.g., violin, clarinet, human voice). Sample libraries attempt to avoid this problem by including many audio recordings. One example is recording each note of the piano, pressed with three different velocities, with every combination of pedals; another example would be recording short notes on a trumpet, both with and without a mute. However, creating such collections of recordings is enormously time-consuming (and thus expensive).

Sampling synthesis works best for instruments with single excitation (e.g., piano, drums): Each excitation is synthesized by playing a new audio recording, mixing recordings together to create polyphony. When attempting to create a long sustained line, some form of “stitching” must be used to combine the recordings from two distinct notes. The simplest such technique is cross-fading: The ending of the first audio recording is gradually reduced down to 0, while the beginning of the second audio recording is gradually amplified up to full volume. This is suitable for percussive instruments and even distinct notes (ideally played *staccato* in music terminology) in string, wind, and brass instruments, but the results are not believable when trying to create slurs (i.e. smoothly connecting two or more notes).

Traditional sampling synthesis has a fixed rule for selecting which segments of recorded audio to use. One alternative to this approach is corpus-based concatenation synthesis, which has no fixed rule. Rather, when synthesizing new audio, the computer must select units from a database of recorded audio with no manually annotated data. This requires automated signal analysis tools, a unit selection algorithm (often based on minimizing a distance function), and optionally various types of transforms to modify the selected units in order to mesh better with other selected units. An overview of this process is given in (Schwarz 2007).

Spectral Modelling Synthesis

Spectral modelling synthesis (SMS) (Serra 1989) is an extremely powerful technique for sound synthesis. Similar to a phase vocoder, it operates by splitting audio into a deterministic portion (sinusoids) and a stochastic portion (filtered white noise). After the relevant number of sinusoids have been

⁴Note that unlike the typical engineering definition of the term “sample” to mean “one discrete data point in a signal”, in the context of electro-acoustic music and sampling synthesis, a “sample” can refer to any amount of audio (generally between 0.01–100 seconds, which would be 441–4,410,000 data points with the customary sampling rate for audio). This confusion of terms is very regrettable, yet highly entrenched in the music industry.

analyzed, they are re-synthesized and subtracted from the recorded audio, leaving a residual signal. The residual is considered to be filtered white noise with time-varying filter parameters, which are then estimated. Musical transformations (i.e. pitch-shifting or time-stretching) can be applied to the sinusoids and time-varying white noise filter, then the results are synthesized. This technique is the basis of the Vocaloid singing synthesis discussed in Section 1.1.2.

Similar to traditional sampling synthesis, the synthesis of a specific note with SMS always uses the same sinusoids+residual data from the annotated data. Due to the stochastic filtered white noise, synthesizing the same note multiple times will produce slightly different audio data, but this amount of randomness is fairly limited. One extension to this idea is to automatically select portions of recorded data (Schoner, Cooper, Douglas & Gershenfeld 1999). Given a database of recorded audio data with additional signals for bow-bridge distance, bow pressure, bow velocity, and finger position, the computer uses machine learning (cluster-weighted modelling, using Gaussian basis terms for probability density estimation) to predict the spectral data (harmonic frequencies and amplitudes) in the desired output audio.

Machine learning was also used to predict spectral data in (Lindemann 2007), although he used a neural network to predict the time-varying frequency and amplitudes of harmonics rather than cluster-weighted modelling. With the exception of particularly noisy portions of the sound (i.e. breath noises or the bow scraping against the string during note attacks), data is stored in a “residual pitch, loudness, harmonics, and noise” form. The data is manually annotated to indicate the type of note transitions (e.g., slurred, tongued) and phrase boundaries; when synthesizing new music, the computer attempts to find phrase boundaries in the written music and then uses fuzzy matching to find the most similar phrase from the database.

Rather than using neural networks to predict spectral data in music based on notation, (Pérez 2009) used neural networks to predict spectral data based on physical data. Pérez constructed a detailed and non-intrusive system for measuring violin performance data (e.g., bow velocity, bow-bridge distance, bow force, bow tilt). Rather than predicting the energy of each harmonic independently, he predicted the energy of each harmonic relative to the overall RMS energy, then used that as another input element to predict the relative energy in each harmonic. Such a system requires a method of generating physical gestures for a (virtual) violinist (Maestre 2009). In addition to non-intrusive measurements of violinists, Maestre constructed Bézier curves which matched the time-varying violin performance data, then performed statistical modelling of those curves to find a best-fit match to the musical score. The resulting performance data was tested with digital waveguide synthesis from the Synthesis Toolkit in C++ (Scavone & Cook 2005) and with Pérez’s SMS-based violin synthesis.

1.2.4 Control of musical synthesis and instruments

There is a great deal of research on controlling music synthesis using a general definition of “control” (i.e. any human interaction with a machine or computer). However, I will limit my discussion to research using the engineering definition of “control”: Adjusting the behaviour of systems with autonomous means. The most common feedback mechanism is a PID (proportional-integral-derivative) controller. There are various methods for tuning the constants in PID controllers; one such heuristic is (Ziegler & Nichols 1942).

One of the earliest frameworks for physical modelling was MOSAIC (Morrison & Adrien 1993), which allowed programmers to combine virtual objects (e.g., strings, bells, acoustic tubes)

with various connection types (e.g., a bow, reed, strike) and controllers (e.g., reading values from a MIDI file, a sine generator, user-programmed functions via Scheme (a Lisp dialect)). The result is synthesized using modal synthesis.

(Cook 1995) presents an overview of a system consisting of a series of modules for a virtual instrument and a virtual musician. The instrument is implemented in two stages: The physics for the instrument family, and the physics of the specific instrument. The performer is separated into three stages: The physics of the performer (e.g., limits on finger speed, maximum derivative of arm movement), expert knowledge of the instrument, and audio perception. Cook's example implementation of a trombone is not impressive by modern standards (the system's "expert knowledge" is merely a lookup table), yet he anticipated that such systems could include neural networks.

Feedback control has been used to control acoustic instruments. (Berdahl, Niemeyer & Smith 2008) investigates a wide range of controllers in terms of mathematical convergence, simulated sound, and finally implementation on an electromagnetically-prepared piano, monochord, and one string guitar. Custom actuators and sensors were designed to minimize non-linearity. (Boutin & Besnainou 2008) added two actuators and an accelerometer to a violin bridge in order to alter the frequency behaviour of the bridge using active control.

Haptic interfaces for computer music

Humans use a great deal of feedback from instruments in order to alter their physical actions to improve the resulting sound. In addition to audio feedback, humans playing real instruments benefit a great deal from haptic (vibrotactile) feedback.

This has been of interest to music technology researchers for decades. (Chafe 1993) noted the importance of such feedback, gave an overview of tactile response (frequency response from 0 to approximately 1000 Hz), and performed a qualitative experiment wherein musicians controlled the lip tension of a physical model of brass instruments by depressing a metal bar. Subjects found it much easier to perform music with the physical model when the metal bar vibrated according to the music.

The vBow (Nichols 2003) is a custom-built virtual violin bow controller. An acrylic bow attached to servomotors provides 4 DOF (lateral, rotational, vertical, and longitudinal motion). A musician may manipulate the bow, which feeds control data to digital waveguide synthesis, which creates audio and force data, which is fed back to the bow through the servomotors.

Increasing processing power allows the haptic feedback to run at audio sample rates. (Sinclair, Florens & Wanderley 2010) investigates haptic interaction via a joystick for two violin physical models: Modal synthesis (running at 44.1 kHz) and a digital waveguide (running at 24 kHz).

Human touch in glabrous (non-hairy) skin is detected with four types of fibres with different frequency sensitivities (Bolanowski, Gescheider, Verrillo & Checkosky 1988). The threshold of detection for frequencies between 0.4 Hz to 500 Hz was measured. The traditional rule of thumb is that 1000 Hz is the upper limit of tactile perception, but this is usually measured with sine waves at a single point of contact with the skin. (Wyse, Nanayakkara, Seekings, Ong & Taylor 2012) tested higher frequencies with the whole hand resting on a vibrating surface and found that sine waves could be detected at 2000 Hz while more complicated signals could even be detected at 4000 Hz. However, the threshold of detection was much lower at high frequencies (100 dB between threshold of detection for FM synthesis at 250 Hz and 2000 Hz).

1.2.5 Music information retrieval

Music information retrieval (MIR) refers to extracting musical information from symbolic data (e.g. sheet music) or signals (e.g., audio or tactile), generally through digital signal processing (DSP) algorithms and machine learning.

Digital signal processing

One of the most useful algorithms in digital signal processing is the Fourier Transform, along with the Fast Fourier Transform (FFT) and Discrete Fourier Transform (Cooley & Tukey 1965). Other than the normal use of FFTs to analyze the frequency components of signals, I benefit from using FFTs for convolution. Convolution is very useful for applying experimentally-measured FIR filters with many coefficients, but a naive implementation of convolution is very slow. (Stockham 1966) showed that convolution could be performed by splitting the input signal into distinct rectangular buffers, multiplying the FFT of each buffer with the FFT of the kernel, taking the inverse FFT, then summing the results (overlap-add).

Since FFTs are central to so many DSP algorithms, it makes sense to optimize their calculation by using a high-quality library such as FFTW (Frigo & Johnson 2005). To use this library, the programmer first calls a “planner”, where the library is notified about the “shape” (size, dimensions, and memory layout) of the problem to solve. FFTW then performs a few tests with various different implementations of FFT algorithms, and selects the fastest implementation. Since most uses of FFTs involve many transforms performed on the same location of memory, adding an “up-front” cost in exchange for faster subsequent calls is a good trade-off.

The field of control theory (Warwick 1989) concerns the behaviour behaviour of a system (digital or analog). This is useful for this dissertation to investigate some problematic behaviour of the physical model in Section 2.2.3.

There is growing concern in the field of DSP research about the reproducibility of research papers (Vandewalle, Kovacevic & Vetterli 2009). The descriptions of algorithms in conference and journal papers are often not sufficiently detailed to allow experts in the field to re-implement them; even if the algorithm can be implemented, certain parameters may have been omitted from the paper, or the dataset(s) of media may not be available. There are various factors which contribute to this state of affairs, such as page limits for academic publications, a desire to avoid overwhelming readers with details, and reproducibility not being perceived as being of importance to paper referees.

Audio analysis

Marsyas (Tzanetakis 2002, Tzanetakis 2007) is an open-source library which provides both DSP and machine learning algorithms, and is widely used in the MIR research community. Marsyas is written in C++, with bindings for python to enable rapid prototyping. The main use of Marsyas is to describe a data-flow network: For example, a network may begin with a sound file, whose samples are sent to various time-domain and spectral-domain feature extractors, whose outputs are fed into a machine learning classifier which judges certain aspects of the audio or else simply saved to a file for use in other machine learning software. An excellent list of widely-used features (most, but not all, supported by Marsyas) is given in (Peeters 2004). A good overview of audio signal processing specifically focused on music is presented in (Klapuri & Davy 2006).

Pitch is a fundamental property of human perception of music, yet detecting it automatically is a surprisingly challenging task. Pitch detection is not the same as fundamental frequency (f_0) detection: Human auditory perception will occasionally perceive a “pitch” which is not present in the audio signal. Well-known examples of this phenomenon are tubular bells and low pitches transmitted via mobile phones. Both cases involve a “missing fundamental”: Consider a tubular bell which we perceive to produce a pitch of 100 Hz. The bell actually does not vibrate at 100 Hz, but instead vibrates at 200 Hz, 300 Hz, 400 Hz, etc. Our auditory perception “fills in” the missing fundamental of 100 Hz. Mimicking this, and other human auditory “tricks”, is much more complicated than f_0 detection.

One of the most successful methods to date is the YIN pitch detection algorithm (de Cheveigné & Kawahara 2002), which is a time-domain algorithm based loosely on autocorrelation. However, instead of multiplying samples, YIN squares the difference between samples and tracks the cumulative mean normalized difference. One drawback of YIN is that is relatively slow, but (Brossier 2006) extended algorithm to use FFTs to perform the autocorrelation.

Although pitch detection is generally more useful for music, detecting frequencies is more useful for measuring raw physical phenomena. One popular method of this is the quadratically interpolated FFT (Abe & Smith 2004). A buffer of audio data is windowed, transformed with the FFT, and peaks are found. However, rather than taking the bin number of a specific peak directly, we instead examine the magnitudes of the peak and the two bins on either side. These three points define a parabola, which improves the accuracy of the estimate of the frequency and magnitude. (Smith 2011) further suggests using a Gaussian window, since a Gaussian transform is a parabola on the log scale.

One additional consideration is how human will perceive the audio. Unless specifically programmed to avoid doing so, computer analysis of digital signals will give equal weight to all frequencies in the signal. However, human auditory sensitivity between 30 Hz and 15000 Hz varies by up to 80 dB (Suzuki & Takeshima 2004). Some audio analysis attempts to mimic this sensitivity by applying a filter with similar frequency response to the input signal.

Another aspect of human perception concerns the onset of a note. A note with an abrupt attack (such as a drum or piano) can be localized in time fairly well (within 10-20 ms), but a note with a soft attack (such as a clarinet or cello playing *piano* and *espressivo*) has a much wider range (50-100 ms) of possible perceived onset times (Wright 2008).

Machine learning and artificial intelligence

Machine learning is the study of computers making judgements about data. A very typical example in music information retrieval is genre classification: Given an unlabelled piece of music (such as an audio CD), the computer judges the audio to be classical, jazz, rock, pop, or heavy metal. This is done through supervised learning, wherein the computer is given some labelled training data (e.g., 100 pieces of music in each musical genre). The computer “learns” how to associate each genre with the underlying audio data (or objective features extracted from that data). When given a piece of unlabelled data, the computer applies its “learning” to produce its best estimate of how a human would label the new data.

One of the most widely-used machine learning algorithms for supervised learning is the support vector machine (SVM) (Boser, Guyon & Vapnik 1992, Cortes & Vapnik 1995). Marsyas uses the popular **libSVM** implementation (Chang & Lin 2011). A trained SVM is the hyperplane between

two sets of data which maximizes the margin between the sets. This is explained in greater detail in Section 5.2.2. The binary classifier can easily be extended to handle multiple independent classes. However, sometimes the classes have a natural order, such as giving preferences on a 5-point scale (e.g., very bad, bad, neutral, good, very good). Treating each rank as an independent class discards potentially valuable information about the ranking, so it would be helpful to use that information during training. This supervised learning problem is called “ordinal regression”. (Li & Lin 2007) discovered a means of reducing ordinal regression to binary classification. This is quite useful, since binary classification has received a great deal of research attention and there are a number of well-optimized computer libraries available (including libSVM). One problem with machine learning is that optimal results require preparing data (normalizing) and selecting certain learning parameters. The authors of libSVM have prepared a very useful guide for practical SVM use, detailed these steps and providing automated steps for finding reasonable learning parameters (Hsu, Chang & Lin 2003).

In addition to machine learning, I use other areas of artificial intelligence, namely optimization and searching a problem space. In particular, I use the simple (yet effective) hill climbing algorithm (Minsky 1961). Given a function $f(x_1, x_2, x_3, x_4, \dots)$, hill climbing attempts to find the maximum value by evaluating the function multiple times with slightly altered inputs, then it “steps” to the set of inputs which produced the highest gradient. This method is sensitive to local maxima, but it is sufficient for my purposes.

Objective analysis of violin sound

Automated classification relies on objective analysis of the target data, yet our perception and judgement of musical sounds are quite subjective. (Wrzeciono & Marasek 2010) attempts to bridge this gap for quality of violins (not violinists). They extracted violin body modes from recorded audio and used the Monte Carlo method to link the audio analysis with subjective judgements from expert musicians, with an overall accuracy of 75%.

Analyzing the quality of violinists (not violins) has obvious pedagogical benefits. (Charles 2010) used traditional machine information learning tools, first extracting audio features such as the spectral centroid and mel-frequency cepstral coefficients, then classifying those features with k -nearest neighbour classifiers. Charles achieved 97% accuracy in four-fold cross-validation when classifying long *legato* bow-strokes as being performed by novice or experts, and between 70%–90% accuracy when attempting to detect specific playing fault such as “crunching” or “skating” sounds.

Another project aimed to train computers to recognize the timbre of specific performers playing the same instrument (Chudy & Dixon 2012). A set of recordings from five players was analyzed to extract various spectral information, which was used to train K-Nearest Neighbour classifiers and perform Linear Discriminant Analysis. Depending on the features and training used, this achieved between 78% and 100% accuracy. However, the authors noted that this was using a small dataset (five players), as it is difficult to gather recordings of cellists playing the same instrument.

Another goal of violin analysis is to extract control parameters from audio: Given only the recorded audio, reproduce the violinist’s actions (i.e. string played, finger position, bow-bridge distance, force, velocity, and tilt). (Pérez & Wanderley 2012) attempts to solve this task by training a statistical model to map from sound to violinist actions while using multiple sensors such as those discussed in Section 1.2.1. Once the model is trained, it is used to predict control parameters from only the audio. This is a new field whose accuracy is not yet high enough for applications.

Musicians use a wide range of terms to describe violin sound, of which very few are directly applicable to objective measures. The relationship between these terms was investigated in (Fritz, Blackwell, Cross, Woodhouse & Moore 2012), with an additional experiment performed in an attempt to link the terms “brighter”, “clear”, “harsh”, “nasal”, and “good” with violin string output digitally filtered to modify the effect of the body resonance. A consistent link was found between “brighter”, “clear”, and “harsh” with energy in specific octave bands, but violinists differed in their interpretation of “nasal” and “good”.

1.2.6 Music education and expressive performance

To train the virtual musician, I draw upon certain aspects of teaching humans to play bowed stringed instruments. This training can be split into low-level physical control of the instrument, and matters of musical style and interpretation of sheet music. The latter is a matter of great interest to music researchers; a great deal of work has gone into attempting to create computer programs which can perform music in a human-like manner.

Music education

In Section 1.1.6, I resolved that the system should be trained similarly to a human. This would allow the virtual musician to benefit from the expertise of human musicians. However, if the virtual musician is to learn like a human, it is important to have realistic expectations of such training. In particular, professional violinists and pianists spend an estimated 10,000 hours practicing their instruments by the time they are 20 years old (Ericsson, Krampe & Tesch-Römer 1993). This figure is the total estimated time spent practicing alone since beginning to learn their instrument at age 4-6. Additionally, even violinists studying to become music teachers (a position with much lower demands on musical skill) had practiced for around 4,000 hours over the same period. This point is worth emphasizing: It takes a great deal of effort for humans to become competent musicians, and our current research in artificial intelligence is far from matching humans for general tasks.

One of the most widely-used methods for teaching music to young children is the Suzuki method. A few elements of this method, and some common criticisms of those points, are worth pointing out:

- Students do not practice alone; a parent is expected to be involved in the daily practice.
- Students learn music from a set repertoire (Suzuki 1978a, Suzuki 1978b), which contains more Baroque music than might be expected. Furthermore, some of the performance indications for those Baroque pieces are not stylistically accurate for that period of history.
- The left-hand fingering is given in the sheet music; although Suzuki students do not read sheet music in the beginning, parents ensure that the student follows the printed fingering.
- Students are not expected to give expressive music performances; a “robotic” performance is an acceptable place to start.

It is not my intention to add to the educational debate concerning the Suzuki method. I merely note that it, and particularly its repertoire books, is a very well-known method of teaching violin⁵.

⁵On a personal note, I learned cello with the Suzuki method, and although I was not an official Suzuki teacher, I often used Suzuki cello books when teaching cello to beginners.

One aspect of expressive performance will not be included in this dissertation: Vibrato is a technique which is not taught to string players until they have 3–4 years of experience. This technique is a sinusoidal alteration of left-hand position, approximately 5.5 Hz for both violin and cello (Geringer & Allen 2004).

Expressive performance

The difficulty of “human-like” performance is easily seen in the Musical Performance Rendering Contest for Computer Systems (Katayose, Hashida, De Poli & Hirata 2012). This contest presents researchers with specific pieces of piano music (mostly classical or Romantic, e.g., Mozart, Beethoven, Chopin). Researchers write computer programs which attempt to perform the works autonomously on a disklavier (a computer-controlled acoustic piano with solenoids placed on the piano keys). This contest is fascinating on a technical level, but on a musical level even the best contest entries sound like piano students with only three or four years of experience. A survey of computer systems for expressive music performance is given in (Kirke & Miranda 2009).

Programs at rencon generally fall into two categories: Rule-driven and data-driven. A good example of rule-driven automated performance is Director Musices (Friberg, Colombo, Frydén & Sundberg 2000), which uses performance rules such as “lengthen the last note of a phrase” and “emphasize notes outside of the current key”. These rules can be altered in real-time (Friberg 2006), making it easier to tweak the constants in the rules. A good example of data-driven automated performance is (Widmer, Flossmann & Grachten 2009), which used recordings of Mozart and Chopin piano pieces performed by famous pianists to train a Bayesian model to map between the scores and expressive performances (timing and velocity deviations). This model is then used to predict an expressive performance of other musical scores.

Some researchers view expressive music performance as an interesting addition to the Turing’s imitation game (Turing 1950). Rather than a computer attempting to imitate a human via text communication, the computer would attempt to perform a piece of music in a human-like manner. If a human judge (or set of judges) was unable to distinguish between human-performed and computer-performed music, then a certain milestone would be reached. Naturally, just as the imitation game (or the “Turing test” as it is often called) has attracted a great deal of attention concerning whether it has any meaning (a good survey of responses is given in (Saygin, Cicekli & Akman 2000)), a similar debate has arisen over any potential implications of an expressive musical performance imitation game.

Many musical scores for computer analysis and performance are shared in the MusicXML (Good 2001) format. As the name implies, this is an encoding of score data in XML, which can be parsed and written with normal XML tools. This format is available free of charge.

1.2.7 Programming and implementation

Many software tools and libraries which were used for this project, so I will restrict this section to only the most vital tools. There are a number of guides for effective programming of scientific software; a good summary is presented in (Aruliah, Brown, Hong, Davis, Guy, Haddock, Huff, Mitchell, Plumley, Waugh, White, Wilson & Wilson 2012).

SciPy (Jones, Oliphant, Peterson et al. 2001–) is a collection of open-source libraries for scientific computing in python. Using python’s high-level language features, SciPy provides modules

for scientific and engineering computing such as linear algebra, signal processing, non-linear solvers, statistics, and optimization. Where applicable, SciPy uses existing C or FORTRAN code to do the bulk of calculations. The combination of the “readability” of python code with the robust and fast implementations of scientific computing algorithms is extremely powerful. One caution about python concerns its multithreading: To simplify various low-level implementation details, python has a Global Interpreter Lock which only allows a single thread to operate at once (Beazley 2010).

Although python (with SciPy) is my language of choice for general computing, the final implementation of the physical model was written in C++. I made extensive use of the open-source C++ compilers in the GNU Compiler Collection (Stallman et al. 2012) and LLVM / Clang++ (Lattner 2005). Alternating between compilers gave my code additional testing, and Clang++ has very good warning and error messages. In addition, the Valgrind tool suite (Nethercote & Seward 2007) contains an extremely useful memory checker and extensive profiler. The highly-optimized C++ code, used for the physical modelling and the CPU-limited portions of the intelligent feedback control, was combined with python using the Simplified Wrapper and Interface Generator (SWIG) library (Beazley 2003).

Since a vibrating string’s motion follows an exponential decay, any attempt at simulating this motion over a long period of time will result in very small numbers. The IEEE 754 (IEEE Computer Society 2008) specification for floating-point numbers contains a special representation of tiny floats: “denormalized” or “subnormal” numbers (Goldberg 1991). On modern desktop computers, calculations involving denormalized numbers are often implemented in microcode instead of directly in the silicon, which is much slower (Dooley & Kale 2006). Disabling the use of denormalized floats is strongly recommended for this purpose. This is discussed further in Section 4.2.3.

Finally, Eigen (Guennebaud, Jacob et al. 2010) is a C++ template library for linear algebra. It automatically vectorizes code for a range of CPU instruction sets, and provides a high-level way to handle matrices in C++. This is discussed further in Section 4.3.1. These vectorized instructions mean that multiple pieces of data can be processed at the same time with single instruction, multiple data (SMD) instructions (Franchetti, Kral, Lorenz & Ueberhuber 2005, Intel 2007, Hassaballah, Omran & Mahdy 2008).

1.3 Problem definition revisited

To summarize, my goal is:

Given a machine-readable representation of sheet music, the computer autonomously produces audio and video that sounds as if it was performed by a human.

In order to maximize the potential applications of this work, and in the spirit of open scientific progress, I adopt the following (voluntary) constraints:

Free software: The entire system must be available at no cost, and furthermore, everybody must be legally allowed to modify it (“free” as in “freedom”).

Easily extended: The system should be clearly designed with good documentation, such that a moderately skilled programmer can add new features. Any data required for the system should be as easy to gather, without requiring any expensive equipment or special acoustic environment.

Human-like pedagogy: The system should be trained in a manner similar to training a human student, since most musicians are experienced at teaching humans but are not programmers.

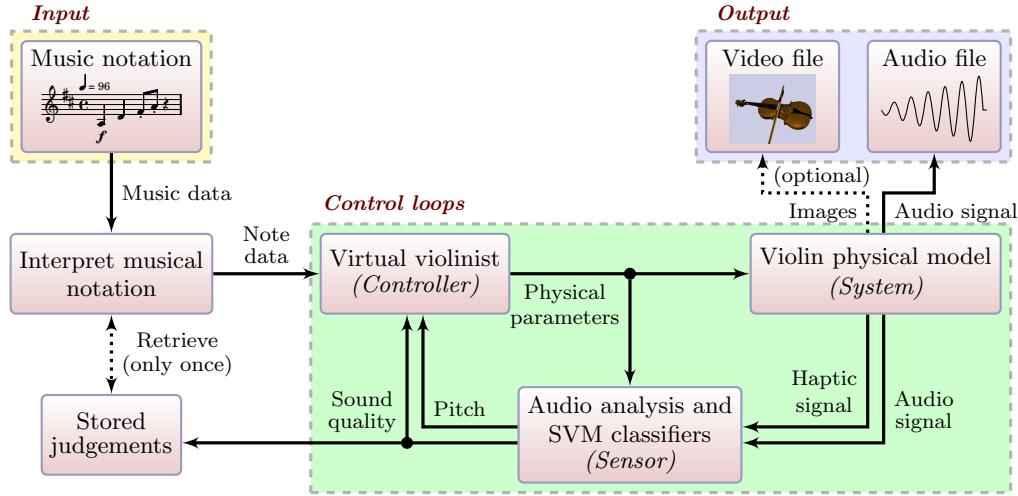


Figure 1.2: Overview of music performance with *Vivi*. We will re-use this diagram throughout this dissertation to add context to each chapter.

1.3.1 Overview of *Vivi, the Virtual Violinist*

Having examined the relevant literature, we can see the overall plan: Given sheet music, translate music notation into physical actions, create audio with a physical model (mathematical simulation) of a stringed instrument, then use feedback control with machine learning to alter the physical parameters if necessary. The sound quality judgements from the machine learning will be stored and used to improve future performances by attempting to avoid making the same mistakes. This process is shown in Figure 1.2.

Following the example of Vocaloid, I named my computer program *Vivi, the Virtual Violinist*, or *Vivi* for short. Each recorded voice in Vocaloid has its own name and illustrated mascot character, which helps software users and video viewers “identify” with the voice (Kenmochi 2010). Naming is a difficult task (Ashworth 1997), but “*Vivi*” will hopefully help musicians feel more at ease with the software. There is no official illustration yet, but the name “*Vivi*” suggests a female violinist, so will be referred to with the feminine pronoun.

To imitate human music pedagogy, human input will happen after each piece of music has finished playing, not in real-time. Music teachers occasionally shout commands or physically adjust music students’ bodies while the student is performing, but this is much less common than giving feedback after the performance is over.

I used the first two Suzuki violin books to train and evaluate *Vivi*, aiming to produce a sound similar to a human violin student playing the same pieces. In the case of Suzuki violin book two, *Vivi* should sound like a 6-year-old violin student. This goal may not seem very ambitious, but I adopt the common idiom “we must learn to walk before we can run”. One of the great advantages of Free software is that we can easily build on each other’s work without losing anything. This dissertation brings the level of autonomous virtual violinists up to a 6-year old child; future work will improve the violinist’s skill level.

1.3.2 Choice of violin physical model

As discussed in Sections 1.2.2 and 1.2.3, there are many methods of computer synthesis of violin-like sounds. However, sampling synthesis and spectral modelling synthesis require huge databases of recorded samples. These can be licensed from expensive commercial software, or recorded from skilled musicians in a sound recording studio — but these options are in conflict with the “Free software” and “Easily extended” constraints. I do not want to rely on resource-heavy methods which would impede other researchers or composers; if a synthesis method would require us to spend a few weeks recording hours of audio data, I consider that a strong point against that method.

The main existing open-source library for physical modelling is the Synthesis Toolkit (STK) in C++ (Scavone & Cook 2005), which uses digital waveguide synthesis. However, the waveguide implementation in STK is based heavily on (McIntyre et al. 1983, Smith 1992) and lacks the recent improvements found in later work⁶.

I therefore decided to write my own physical modelling code, and faced the choice of digital waveguide or modal synthesis. (Demoucron 2008) presented a detailed description of modal synthesis with many sound examples⁷ of the results. Unfortunately, sound examples of waveguide synthesis were difficult to find. There are a few examples from STK available⁸, but as previously noted these represent old implementations and are not indicative of state-of-the-art research in waveguide synthesis. Furthermore, Demoucron’s examples included convolution with the violin body impulse and synthesized four strings separately, while the STK examples synthesized only a single string.

The comparison was not at all valid, but I had to make a choice. A very rough survey of research papers suggested that digital waveguide was used more often than modal synthesis, and various papers (including (Demoucron 2008)) noted that modal synthesis was slower than waveguide synthesis. However, the modal synthesis described by (Demoucron 2008) represented a known quality: I was quite impressed with his audio examples and his clear and thoughtful explanation of the algorithm.

In the end I chose to use Demoucron’s modal synthesis. He does not claim that his algorithm is original or highly advanced, but instead notes that

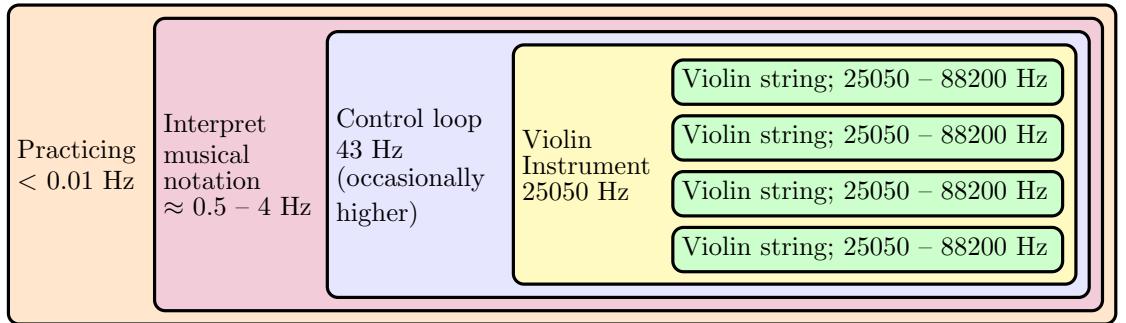
“A main purpose of our work with the bowed-string model was to separate the properties of the model that are sufficient for obtaining an acceptable violin synthesis, from the demands necessary for obtaining a realistic modelling of the mechanics of the bowed string. It should be emphasized that our model has not been developed to be a scientific simulation tool for detailed studies of the bowed string, and in many cases it will not meet the demands of such a tool. However, in the light of our objectives, it is considered to perform perfectly satisfactorily, allowing perceptually convincing simulations of bowed-string sounds.” (Demoucron 2008, p. 74)

Like Demoucron’s work, my objective is not a scientifically-accurate reproduction of stringed instrument physics; rather, I wish to produce acceptable simulations of their sound. This model satisfies my objectives, so I adopt it despite its simplification of certain physical processes. These simplifications are discussed in Section 2.4.1.

⁶In fact, when I began this research, the bowed-string modelling in STK did not even include control of the bow velocity. Bow velocity was added to STK by Esteban Maestre in version 4.4.3, released on 2011 August 30.

⁷<http://recherche.ircam.fr/equipes/instruments/demoucron/>

⁸http://ccrma.stanford.edu/~jos/waveguide/Sound_Examples.html

Figure 1.3: Sampling rates of various parts of *Vivi*.

1.4 Organization of this dissertation

In this chapter, I explained the problem and the constraints, motivated the constraints with examples of potential applications, and examined related work. The remainder of the dissertation is designed in layers, where I begin by discussing the smallest time interval (the violin string) and then gradually expand, focusing on each step in turn. This is illustrated in Figure 1.3.

There are two main parts to the thesis, with an additional introduction, conclusion, and appendices.

Part I: Physical Modelling of the Violin Family

The first part simulates the sound of physical instruments.

Chapter 2, Physics of Physical Modelling: Gives the mathematical formulae which simulates the violin string and instrument body, and discusses the design decisions and consequences of using those equations.

Chapter 3, Constants for Physical Modelling: Discusses the physical experiments and simulations used to select constants to be used with the physical modelling equations.

Chapter 4, Implementation of Physical Modelling: Explains the video generation and the implementation of the physical model as a high-performance C++ library.

Part II: Performing with the Virtual Violin Family

The second part simulates the behaviour of a beginning violinist, violist, and cellist.

Chapter 5, Control loops: Describes the pitch and bow control loops central to the virtual musician with classical and intelligent feedback control.

Chapter 6, Calibration, Performance, and Self-Improvement: Discusses the human involvement in training the virtual violinist, various automated processes to improve the output, and the mapping from musical score to physical actions.

Chapter 7. Implementation of *Vivi, the Virtual Violinist* : Describes the extraction of information from multi-instrument scores and the implementation.

Conclusion

Examines the project with reference to the applications discussed in the introduction, gives a few philosophical remarks about the lack of philosophical implications of virtual musicians, and discusses future directions of this research.

Appendices

These include additional material which may be of interest.

Appendix A, Additional Mathematics for Physical Modelling: This supports a conjecture concerning one bowing variable.

Appendix B, Performances of select sheet music: Audio and video generated automatically from sheet music.

Appendix C, Source code, raw data, and copyleft licenses: All source code and data for this research are available and published under permissive copyright licenses: GPLv3 for source code, and Creative Commons BY-SA 2.5 Scotland for this dissertation.

Part I

Physical Modelling of the Violin Family

Chapter 2

Physics of Physical Modelling

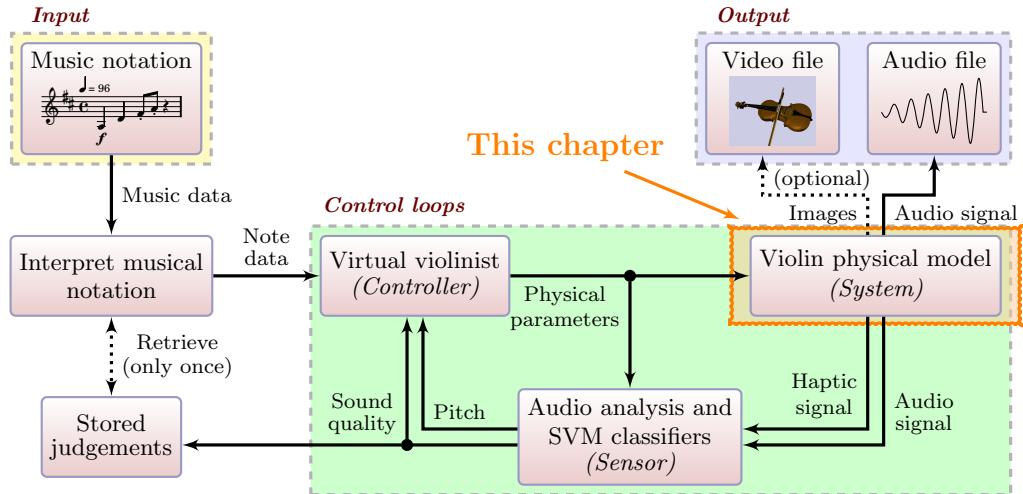


Figure 2.1: Physical modelling in context.

In order to teach a computer how to play musical instruments, I need a model (or “simulation”) of the relevant instruments. In other words, I want to answer questions such as:

If I place a violin bow on the violin D string at 0.16 string-lengths away from the bridge, press down with a force of 1.23 N, and accelerate the bow from rest at 8.0 m/s^2 for 0.023 seconds; what output will I get? Given the current state of the violin string, suppose I increase the bow force to 1.34 N, accelerate at 4.0 m/s^2 , and simulate for another 0.023 seconds; what output will I get?

These questions are answered in this chapter and the next two: Chapter 2 discusses the physical modelling algorithm, Chapter 3 discusses experimental measurements of real-world instruments and simulated experiments to determine constants which improve the model’s output, and Chapter 4 discusses the computer implementation of this algorithm and the video production.

In Section 1.2.2 and Section 1.3.2, I examined various physical models of a violin, and selected (Demoucron 2008) as being the best fit for my goals. In this chapter, I extended Demoucron’s algorithm to include plucking the string, improved the bowing friction model, and added haptic output. Formally, the model takes five input parameters (i.e. the physical actions) and produces two output signals (shown in Figure 2.2 and Table 2.1). To allow some computer optimizations, each set of

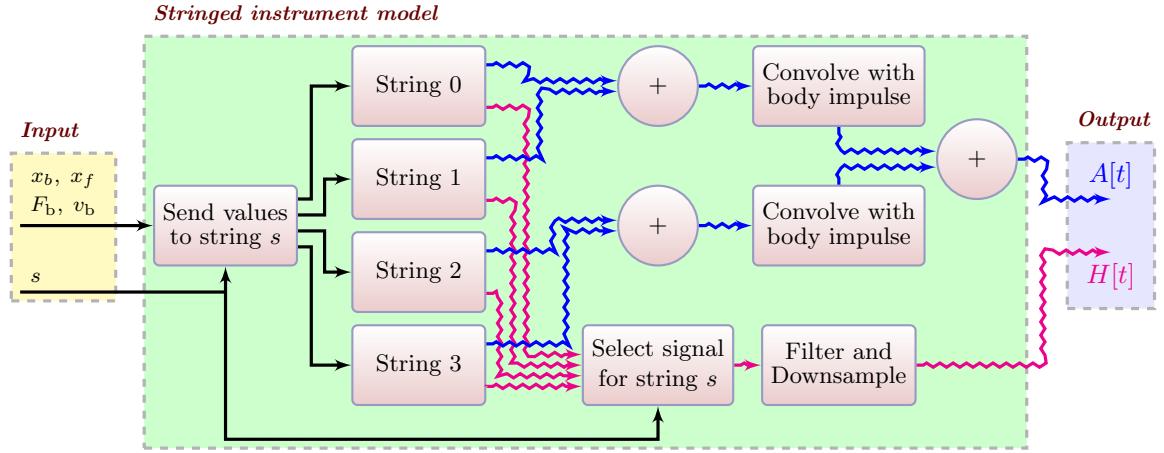


Figure 2.2: Overview of the violin physical modelling. Solid lines indicate transfer of variables; wavy lines indicate signals. Each set of input parameters are generally evaluated for 512 samples. Note that each string continues to produce audio after the bow moves to a different string, whereas the bow force comes from only the current string.

Symbol	Explanation	Symbol	Explanation
x_b	Bow-bridge bowing distance [m]	$a_n(t)$	Modal displacement
x_f	Finger position [m]	$\dot{a}_n(t)$	Modal velocity
v_b	Bow velocity [m/s]	$\phi_n(x)$	Eigenvectors
F_b	Bow force [N]	F_i	External forces
s	String number [0, 1, 2, 3]	D_i	Coefficients for force calculations

(a) Input variables	(b) Main internal variables						
<table border="1"> <thead> <tr> <th>Symbol</th> <th>Explanation</th> </tr> </thead> <tbody> <tr> <td>$A[t]$</td> <td>Audio output [16-bit signal at 22050, 44100, or 66150 Hz]</td> </tr> <tr> <td>$H[t]$</td> <td>Haptic response at bow hand [16-bit signal at 22050, 44100, or 66150 Hz]</td> </tr> </tbody> </table>	Symbol	Explanation	$A[t]$	Audio output [16-bit signal at 22050, 44100, or 66150 Hz]	$H[t]$	Haptic response at bow hand [16-bit signal at 22050, 44100, or 66150 Hz]	(c) Output variables
Symbol	Explanation						
$A[t]$	Audio output [16-bit signal at 22050, 44100, or 66150 Hz]						
$H[t]$	Haptic response at bow hand [16-bit signal at 22050, 44100, or 66150 Hz]						

Table 2.1: Main variables used in bowed-string algorithm. Since the model will be used for violin, viola, and cello, there is implicitly a sixth parameter: The instrument to model, and therefore which constants to use. However, this parameter does not change during the simulation, so we omit it from this list of variables. The sampling rates for $A[t]$ and $H[t]$ here refer to the sample rate within the string simulation; the instrument simulation decimates those to 22050 Hz and 11025 Hz respectively.

input parameters are held constant for 512 samples, but if a musical note boundary requires a smaller buffer, this is reduced. The instrument is simulated at a constant 22050 Hz, while the sampling rate of individual strings vary.

This chapter can be divided into three sections:

1. In Section 2.1, I describe the equations used for the stringed instrument physical modelling algorithm.
2. Additional discussion of the model's behaviour and trade-offs of accuracy vs. speed are given in Section 2.2 for finger actions and Section 2.3 for bowing actions.
3. Final remarks are given in Section 2.4, giving a summary of external actions the model can simulate. Possible improvements to the model are discussed.

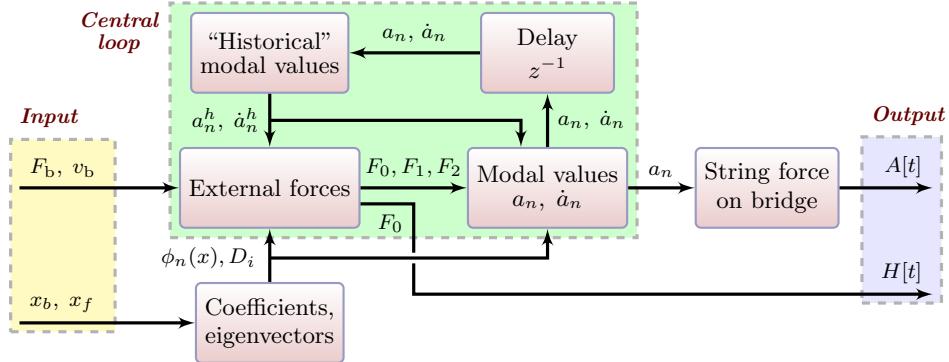


Figure 2.3: Overview of the violin string physical modelling. Variables with a subscript n indicate an N-element vector. Note that if x_b and x_f have not changed, we can avoid recomputing ϕ_n .

2.1 String and instrument body simulation

The central part of a stringed instrument model is of course the actual string simulation; an overview is given in Figure 2.3 and Table 2.1.

The calculation of the wave equation in Section 2.1.1 and the first portion of the bow force calculations in Section 2.1.3 are unchanged from Demoucron's algorithm; for longer explanations, proofs, and analysis of this portion of the model's behaviour, see (Demoucron 2008) and/or (Demoucron & Rasamimanana 2009). To permit easy cross-referencing, I have used the same symbols and terminology as Demoucron's work. However, there are two warnings about these variables. First, he uses f_n to represent the magnitudes of modal forces, which conflicts with the customary usage in digital signal processing of f_n being the frequency of the n^{th} mode. This notation is unfortunate as I need to discuss both modal forces and frequencies of modes, so I use \check{f}_n to refer to the magnitudes of modal forces. Second, he uses a subscript n to indicate a vector, such as $a_n = a_n^h + X_{3n}\check{f}(t_1)$. Some scientists and mathematicians may prefer to express this in the more compact form $\mathbf{a} = \mathbf{a}^h + \mathbf{X}_3 \circ \check{\mathbf{f}}(t_1)$.

The overall sampling rate is $f_s = 22050$ Hz, but the sampling rate of each string is set to be a multiple of 1, 2, or 4 times the overall sampling rate, based on balancing output quality and processing time for each string. The output of strings of the same frequency multiple are summed together, convolved with the combined body response and low-pass filter, decimated, then summed with the output of strings operating at a different frequency multiple. More details are given in Section 2.1.5.

Before discussing the physical modelling, I will discuss the expected range of input parameters. During normal playing by experts, (Schoonderwaldt 2009) found that the violin bow-bridge distance x_b ranged from 5 mm to 60 mm, the bow velocity v_b ranged from 0.05 m/s to 1.0 m/s (with these values being negative for upbows), while the bow force F_b was between 0.1 N to 2 N.

I made some rough estimates on beginner violin and cello playing. On the violin, x_b was between 20 mm and 70 mm, while on cello x_b was between 35 mm and 110 mm. The bow velocity on both instruments went up to 0.5 m/s. Bow force is harder to observe, so I refrained from making any wild estimates. Finally, the highest note in Suzuki violin books 1 and 2 is the 4th finger on the E string, occurring at $x_f = 0.333L$ ($x_f = 110$ mm) and having a fundamental frequency of 990 Hz.

2.1.1 String physics

The synthesis begins with a stiff string with linear density ρ_L , tension T , Young's modulus E , diameter d , length L , and second moment of area for a circular cross-section of the string $I = \frac{\pi d^4}{64}$. We also add a damping coefficient $R_L(\omega)$ to cover all losses of energy: Losses along the string due to friction, the wave reflection at both ends of the string, and energy transferred to the instrument body. With external forces $F_i(x, t)$ and transverse displacement $y(x, t)$ of the string at position x and time t , the wave equation is:

$$\rho_L \frac{\partial^2 y(x, t)}{\partial t^2} - T \frac{\partial^2 y(x, t)}{\partial x^2} + EI \frac{\partial^4 y(x, t)}{\partial x^4} + R_L(\omega) \frac{\partial y(x, t)}{\partial t} = \sum_i F_i(x, t) \quad (2.1)$$

For a modal solution in terms of eigenvectors $\phi_n(x)$, the following substitutions are made. These substitutions are only strictly valid as N tends to ∞ ; for finite N they result in approximations.

$$\phi_n(x) = \sqrt{\frac{2}{L}} \sin\left(\frac{n\pi x}{L}\right) \quad (2.2)$$

$$a_n(t) = \int_0^L \phi_n(x) y(x, t) dx \quad y(x, t) = \sum_{n=1}^N \phi_n(x) a_n(t) \quad (2.3)$$

$$\check{f}_n(t) = \int_0^L \phi_n(x) F(x, t) dx \quad F(x, t) = \sum_{n=1}^N \phi_n(x) \check{f}_n(t) \quad (2.4)$$

The wave equation (2.1) can be rewritten in modal form,

$$\ddot{a}_n(t) + 2r_n \dot{a}_n(t) + w_{0n}^2 a_n(t) = \rho_L^{-1} \check{f}_n(t) \quad (2.5)$$

$$r_n = \frac{R_L(\omega)}{2\rho_L} \quad \omega_{0n} = \sqrt{\frac{T}{\rho_L} \left(\frac{n\pi}{L}\right)^2 + \frac{EI}{\rho_L} \left(\frac{n\pi}{L}\right)^4} \quad (2.6)$$

Experimental measurements (Section 3.2) will provide r_n directly, giving the damped frequencies

$$\omega_n = \sqrt{\omega_{0n}^2 - r_n^2} \quad (2.7)$$

For simplicity, external forces are assumed to be constant throughout our time interval dt and each force acts on a single point. We can therefore represent each force with a Dirac delta function and update the modal equation (2.5),

$$\ddot{a}_n(t) + 2r_n \dot{a}_n(t) + w_{0n}^2 a_n(t) = \rho_L^{-1} \sum_i F_i(t) \phi_n(x_i) \quad (2.8)$$

The modal impulse response from (2.8) is

$$h_n(t) = \omega_n^{-1} \sin(\omega_n t) e^{-r_n t} \quad (2.9)$$

The modal displacements at time $t_1 = t_0 + dt$ with modal forces $\check{f}_n(t)$ are:

$$a_n(t_1) = X_{1n} a_n(t_0) + X_{2n} \dot{a}_n(t_0) + \frac{1}{\rho_L} \int_{t_0}^{t_1} \check{f}_n(t') h_n(t_1 - t') dt' \quad (2.10)$$

$$\begin{aligned}
X_{1n} &= \left(\cos(\omega_n dt) + \frac{r_n}{\omega_n} \sin(\omega_n dt) \right) e^{-r_n dt} & Y_{1n} &= - \left(\omega_n + \frac{r_n^2}{\omega_n} \right) \sin(\omega_n dt) e^{-r_n dt} \\
X_{2n} &= \frac{1}{\omega_n} \sin(\omega_n dt) e^{-r_n dt} & Y_{2n} &= \left(\cos(\omega_n dt) - \frac{r_n}{\omega_n} \sin(\omega_n dt) \right) e^{-r_n dt} \\
X_{3n} &= \frac{1 - X_{1n}}{\rho_L \omega_{0n}^2} & Y_{3n} &= \frac{-Y_{1n}}{\rho_L \omega_{0n}^2}
\end{aligned}$$

Table 2.2: Coefficients for modal displacement and velocity.

with coefficients for the modal displacements and velocities given in Table 2.2. The calculation of each time step begins by calculating the new modal displacements and velocities which would occur if no external forces were applied¹. Intuitively, we are asking “How would the string behave if there was no finger or bow on the string?” Concretely, we define

$$a_n^h(t_1) = X_{1n}a_n(t_0) + X_{2n}\dot{a}_n(t_0) \quad (2.11)$$

$$\dot{a}_n^h(t_1) = Y_{1n}a_n(t_0) + Y_{2n}\dot{a}_n(t_0) \quad (2.12)$$

Since the model assumes that $\check{f}_n(t)$ has a constant value throughout our time interval dt , and

$$\int_{t_0}^{t_1} \check{f}_n(t') h_n(t_1 - t') dt' = \frac{1}{\omega_{0n}^2} (1 - X_{1n}) \check{f}_n(t_1) \quad (2.13)$$

we can simplify (2.10) with (2.11), and apply a similar reasoning to the modal velocities,

$$a_n(t_1) = a_n^h(t_0) + X_{3n}\check{f}_n(t_1) \quad (2.14)$$

$$\dot{a}_n(t_1) = \dot{a}_n^h(t_0) + Y_{3n}\check{f}_n(t_1) \quad (2.15)$$

Depending on the actions of the violinist, the model will include up to 3 external forces (F_0 , F_1 , F_2). Each force acts at a single distinct point on the string. The calculation for the forces are given in Section 2.1.2 and Section 2.1.3. Once these forces are calculated, the modal forces are

$$\check{f}_n(t_1) = \sum_i \phi_n(x_i) F_i(t_1) \quad (2.16)$$

Once $a_n(t)$ has been calculated, the force of the bow on the bridge produces the audio signal. G_n can be pre-computed to save time during simulation:

$$\begin{aligned}
F_{\text{bridge}}(t) &= T \frac{\partial y(x, t)}{\partial x} \Big|_{x=0} - EI \frac{\partial^3 y(x, t)}{\partial x^3} \Big|_{x=0} \\
&= \sum_{n=1}^N a_n(t) G_n, \quad \text{where } G_n = \sqrt{\frac{2}{L}} \left(T \left(\frac{n\pi}{L} \right) + EI \left(\frac{n\pi}{L} \right)^3 \right)
\end{aligned} \quad (2.17)$$

¹To maintain a consistent notation with (Demoucron 2008), we use the h superscript which he used to mean “historical”. However, we suggest that h be understood as “human-free” or “hands-free”.

External forces on the string

The forces applied from external actions will be modelled as damped springs and the bow-string friction equation. These in turn will benefit from knowing the “hands-free” displacement and velocity at certain points along the string. To shorten the equation, I define $y_i = y(x_i, t)$ and $v_i = \dot{y}(x_i, t)$.

$$y_i^h = \sum_{n=1}^N \phi_n(x_i) a_n^h \quad v_i^h = \sum_{n=1}^N \phi_n(x_i) \dot{a}_n^h \quad (2.18)$$

It will also be useful to calculate the actual displacement and velocity at certain points, given external point forces F_j :

$$A_{pq} = \sum_{n=1}^N \phi_n(x_p) \phi_n(x_q) X_{3n} \quad B_{pq} = \sum_{n=1}^N \phi_n(x_p) \phi_n(x_q) Y_{3n} \quad (2.19)$$

$$y_i = y_i^h + \sum_j A_{ij} F_j \quad v_i = v_i^h + \sum_j B_{ij} F_j \quad (2.20)$$

Given a damped spring of strength K and damping R , the restoring force F depends on its current displacement y_i , the desired displacement y_i^d , and its velocity v_i :

$$F_{\text{damped}}(t) = -K(y_i - y_i^d) - Rv_i \quad (2.21)$$

2.1.2 Finger forces on the string

The most common use of a violin physical model is to simulate bowing actions, but I first consider the two simpler cases of external actions on the violin: Plucking the string, and pressing a left-hand finger on the string. To avoid certain problems discussed in Section 2.2, the finger actions are modelled as two or three damped springs (forces F_0, F_1, F_2). It should be clarified that these are *transverse* forces: if the x -axis is along the string and the y -axis is along the direction of the bow, a violinists’ finger presses down along the z -axis, but these forces are the effects of the finger along the y -axis.

As shown in Figure 2.4, the position x_1 of force F_1 always represents the edge of the left-hand finger closest to the bridge. The positions of the other two forces vary based on the violinist’s action, with x_p and x_f being the positions of the pluck and finger. Other constants are $W_{p,f}$ being the widths, $K_{p,f}$ being the spring constants, and $R_{p,f}$ being the damping constants, discussed in Section 3.5.3.

Plucking the string

F_0 and F_2 represents the two sides of the plucking finger, which will cause the string to be pulled to one side for some time until the string is released. F_1 represents the left-hand finger. The desired displacement of the “plucking points” x_0 and x_2 is y_p^d , with $(y_0^d = y_2^d = y_p^d)$. At the beginning of a pluck, $y_p^d = 0$ and increases at speed v_p until it reaches the desired displacement y_p^d . This position is held for the desired duration t_p , at which time the string is released.

$$\begin{aligned} x_0 &= x_p & x_1 &= x_f & x_2 &= x_p + W_p \\ K_0 &= K_p & K_1 &= K_f & K_2 &= K_p \\ R_0 &= R_p & R_1 &= R_f & R_2 &= R_p \end{aligned} \quad (2.22)$$

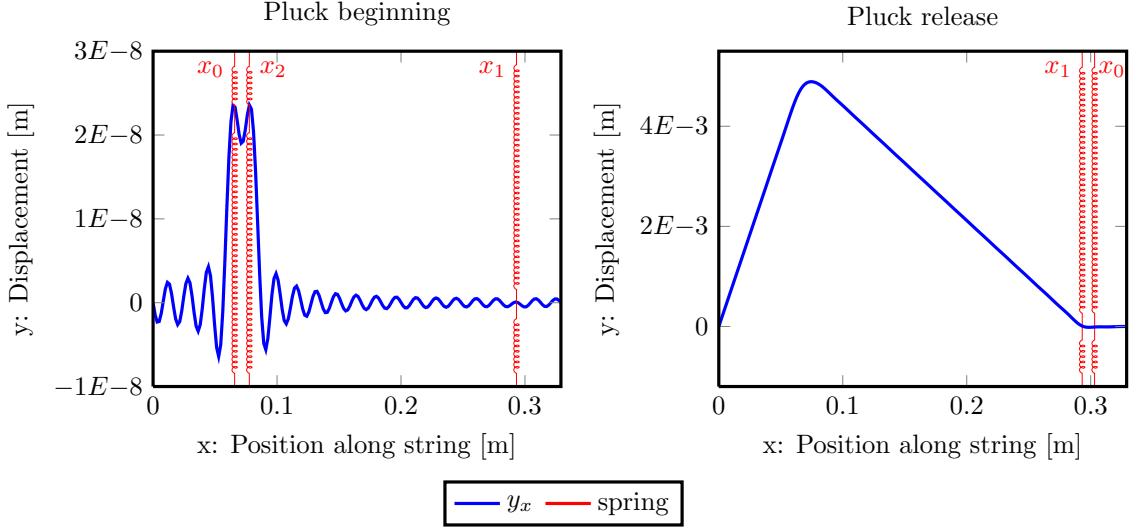


Figure 2.4: String displacements at beginning and release of pluck on the violin G string, $x_f = 0.891L = 0.29$, $x_p = 0.2L = 0.066$. Left: beginning of pluck. Right: releasing the string. The “wiggles” in the left graph are unwanted artifacts due to the finite number of modes ($N = 40$). Note that the y-axes show different ranges, since the string moves only slightly in the first time sample; although the “wiggles” are not desirable they are not a serious problem. The violin bridge is at position $x = 0$, while the nut is at $x = 0.329$.

Consider the combination of forces and displacements at points x_0 , x_1 , and x_2 by combining (2.21) with (2.20). Note that since x_1 always represents the finger, its equation does not include a y_p^d term.

$$\begin{aligned} (A_{00}K_0 + B_{00}R_0 + 1)F_0 + (A_{01}K_0 + B_{01}R_0)F_1 + (A_{02}K_0 + B_{02}R_0)F_2 &= -K_0(y_0^h - y_p^d) - R_0v_0^h \\ (A_{10}K_1 + B_{10}R_1)F_0 + (A_{11}K_1 + B_{11}R_1 + 1)F_1 + (A_{12}K_1 + B_{12}R_1)F_2 &= -K_1(y_1^h) - R_1v_1^h \\ (A_{20}K_2 + B_{20}R_2)F_0 + (A_{21}K_2 + B_{21}R_2)F_1 + (A_{22}K_2 + B_{22}R_2 + 1)F_2 &= -K_2(y_2^h - y_p^d) - R_2v_2^h \end{aligned} \quad (2.23)$$

These three linear equations can be expressed in matrix form:

$$\begin{aligned} \bar{A}z &= b \\ \bar{A} &= \begin{bmatrix} A_{00}K_0 + B_{00}R_0 + 1 & A_{01}K_0 + B_{01}R_0 & A_{02}K_0 + B_{02}R_0 \\ A_{10}K_1 + B_{10}R_1 & A_{11}K_1 + B_{11}R_1 + 1 & A_{12}K_1 + B_{12}R_1 \\ A_{20}K_2 + B_{20}R_2 & A_{21}K_2 + B_{21}R_2 & A_{22}K_2 + B_{22}R_2 + 1 \end{bmatrix} \\ z &= \begin{bmatrix} F_0 \\ F_1 \\ F_2 \end{bmatrix} \quad b = \begin{bmatrix} -K_0(y_0^h - y_p^d) - R_0v_0^h \\ -K_1(y_1^h) - R_1v_1^h \\ -K_2(y_2^h - y_p^d) - R_2v_2^h \end{bmatrix} \end{aligned} \quad (2.24)$$

The forces F_0 , F_1 , and F_2 are calculated from (2.24), either by finding the inverse of \bar{A} and solving $z = \bar{A}^{-1}b$, or using a matrix decomposition. Solving (2.24) with either of these techniques is particularly useful for the string model: Although the vector b changes based on the current state of the string, the matrix \bar{A} only depends on the positions of external forces. Since the positions of external actions change infrequently (usually once every 0.5–1.0 seconds), a matrix inverse or decomposition saves a great deal of computation. Further implementation details are given in Section 4.3.1.

$$\begin{aligned}
D_8 &= \frac{-1}{K_f A_{00} + R_f B_{00} + 1} & D_9 &= (K_f A_{01} + R_f B_{01}) D_8 \\
D_{10} &= -(R_f B_{01} + K_f A_{01}) L_3 & D_{11} &= (R_f B_{00} + K_f A_{00} + 1) L_3 \\
L_3 &= \frac{-1}{(A_{00} K_f + B_{00} R_f + 1)(A_{11} K_f + B_{11} R_f + 1) - (A_{01} K_f + B_{01} R_f)^2}
\end{aligned}$$

Table 2.3: Coefficients for pluck release.

String release

F_0 and F_1 represent points on the left-hand finger, and act against any movement ($y_i^d = 0$). F_2 is unused. If there is no left-hand finger on the string, then we set all $F_i = 0$.

$$\begin{aligned}
x_0 &= x_f + W_f & x_1 &= x_f & x_2 &= 0 \\
K_0 &= K_f & K_1 &= K_f & K_2 &= 0 \\
R_0 &= R_f & R_1 &= R_f & R_2 &= 0
\end{aligned} \tag{2.25}$$

These values could be inserted into the three-force equations in (2.24). However, since we only use two forces during the string release, we can simplify \bar{A} and b as

$$\bar{A} = \begin{bmatrix} A_{00} K_f + B_{00} R_f + 1 & A_{01} K_f + B_{01} R_f \\ A_{01} K_f + B_{01} R_f & A_{11} K_f + B_{11} R_f + 1 \end{bmatrix} \quad b = \begin{bmatrix} -K_f y_0^h - R_f v_0^h \\ -K_f y_1^h - R_f v_1^h \end{bmatrix} \tag{2.26}$$

We now define the augmented matrix $M = [\bar{A}|b]$,

$$M = \left[\begin{array}{cc|c} A_{00} K_f + B_{00} R_f + 1 & A_{01} K_f + B_{01} R_f & -K_f y_0^h - R_f v_0^h \\ A_{01} K_f + B_{01} R_f & A_{11} K_f + B_{11} R_f + 1 & -K_f y_1^h - R_f v_1^h \end{array} \right] \tag{2.27}$$

The canonical row-echelon form of a 2x3 augmented matrix is:

$$\left[\begin{array}{cc|c} 1 & \frac{M_{01}}{M_{00}} & \frac{M_{02}}{M_{00}} \\ 0 & 1 & \frac{M_{00}M_{12} - M_{02}M_{10}}{M_{00}M_{11} - M_{01}M_{10}} \end{array} \right] \tag{2.28}$$

Since only the M_{i2} entries of M change at each time sample, the bulk of calculations in (2.28) can be pre-computed as the coefficients defined in Table 2.3,

$$\left[\begin{array}{cc|c} 1 & -D_9 & (K_f y_0^h + R_f v_0^h) D_8 \\ 0 & 1 & (K_f y_0^h + R_f v_0^h) D_{10} + (K_f y_1^h + R_f v_1^h) D_{11} \end{array} \right] \tag{2.29}$$

Solving for F_i with back-substitution gives

$$F_1 = (K_f y_0^h + R_f v_0^h) D_{10} + (K_f y_1^h + R_f v_1^h) D_{11} \tag{2.30}$$

$$F_0 = (K_f y_0^h + R_f v_0^h) D_8 + F_1 D_9 \tag{2.31}$$

The row-echelon form of the 3x4 augmented matrix form of (2.24) can be solved with a computer algebra system, but the coefficients are considerably more complicated than the 2x3 case. Since the pluck release is much longer than the initial pluck, there is little to gain by optimizing that step.

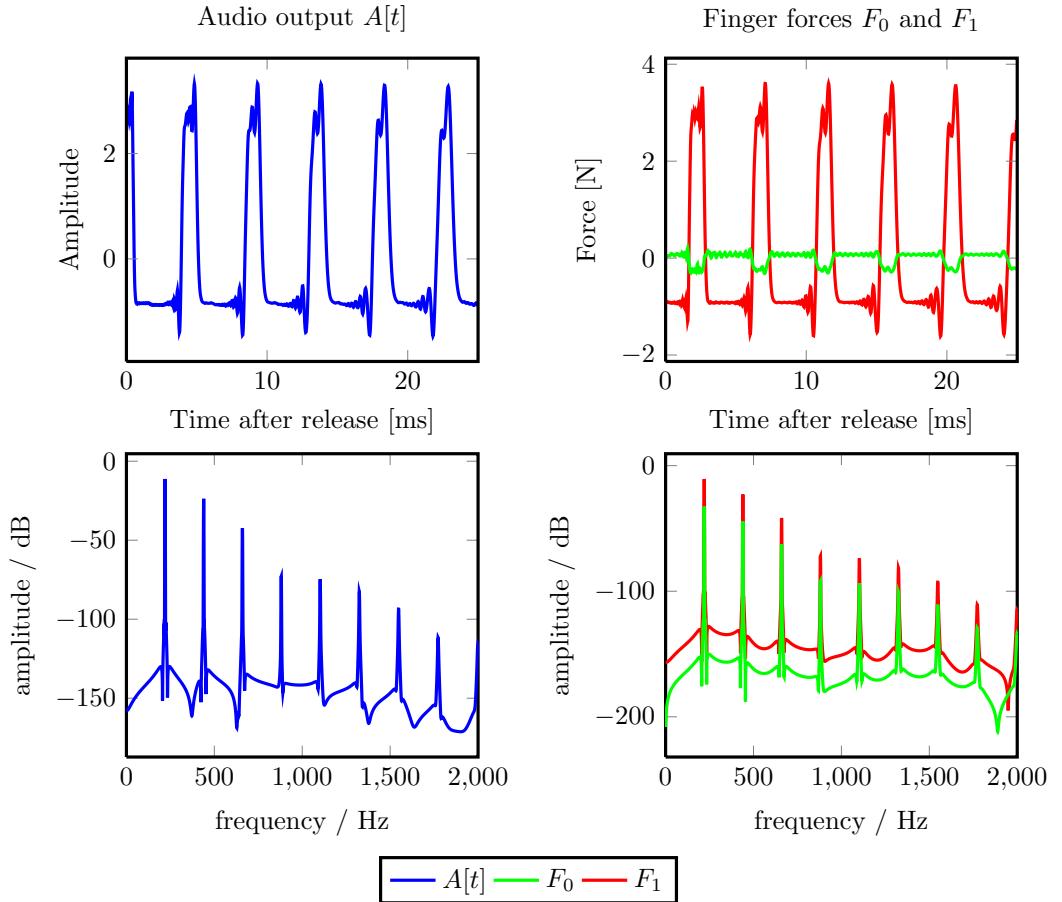


Figure 2.5: Finger forces F_0 and F_1 causing desired pitch ($x_1 = 0.891L$, 220 Hz) after a pluck release. The time-domain plots only show the first few cycles, but the FFT was performed on 65536 samples (≈ 1.5 seconds) after applying a Blackman-Harris window.

Audio 2.1: Plucking a fingered violin G string

<http://percival-music.ca/dissertation/a.2.1.pluck-finger-forces-violin-g.wav>

Pitches and energy transfer between modes

The external forces do not change the fundamental constants of the string: The modal frequencies ω_n do not change due to x_0 , x_1 , or x_2 . Different pitches are produced due to forces F_0 , F_1 , and F_2 transferring energy between modes. Without this modal cross-coupling, the system would be a linear system of decaying modes and would require altering the string length L in order to create different pitches.

Figure 2.5 shows the output of the simulation after a fingered pluck. Forces F_0 and F_1 act to suppress any movement at positions x_0 and x_1 , while the remainder of the string vibrates freely (recall that F_2 is not used for the release portion of the pluck). Musically speaking, the plucking position ($x_p = 0.891L$) corresponds to the normal position of the first finger in violin playing, so on the violin G string we expect to see a full set of harmonics beginning at 220 Hz.

The displacements along the entire string for one cycle is shown in Figure 2.6. That figure also illustrates why F_1 is considerably larger than F_0 , and why the two forces are almost exactly out of phase: F_1 must counter the behaviour of the long vibrating portion of the string, whereas F_0 need only cancel the smaller movements in the short portion of the string.

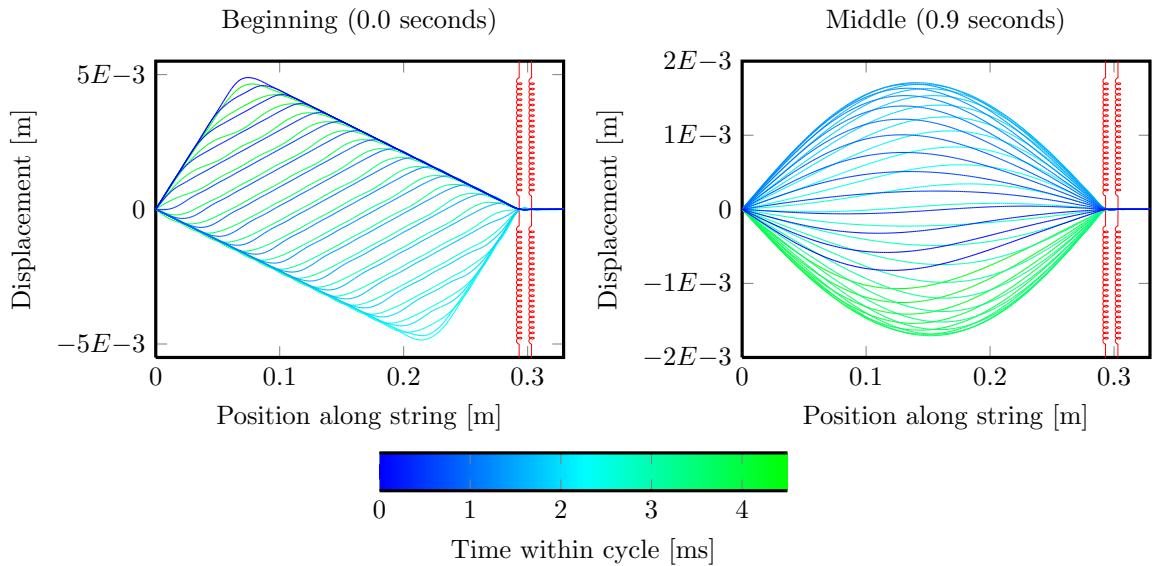


Figure 2.6: String displacements during one cycle of a pluck on the violin G string, downsampled by a factor of 5. Times are measured after pluck release. Note that while the string has very small displacements at points x_0 and x_1 , the string vibrates freely between those points. This is an unwanted artifact of modelling the finger as only two points on the string. Note the string has a sharp “corner” at the plucking position ($x_p = 0.2L = 0.066$) at time 0, and that this “corner” moves to the right until it hits the finger, then returns in the opposite direction, forming a rough parallelogram. However, over time this “corner” becomes more and more gradual (courtesy of the high decay rates of upper modes), until it reaches the arc-like string segments shown on the right.

This simulation produced Audio 2.1.

Video 2.1: Displacements after plucking a fingered violin G string: 4.5 ms, full sample rate.

<http://percival-music.ca/dissertation/v.2.1.pluck-finger-movie-violin-g.mpeg>

Video 2.2: Displacements after plucking a fingered violin G string: 200 ms, reduced sample rate.

<http://percival-music.ca/dissertation/v.2.2.pluck-finger-movie-long-violin-g.mpeg>

Oddities of this model

Although the right-hand plucking finger is always modelled with two forces, the left-hand finger is modelled with one or two forces. It may seem odd to alter the simulated left-hand finger width — when plucking the string, the finger is infinitely thin (at point x_f), whereas when the string is released, the finger acts on the string at two points (x_f and $x_f + W_f$). Furthermore, the “finger” allows free vibrations between those points. Using only three forces for the plucking portion and two forces for the string release is a compromise between computational complexity and realism of the model. Adding additional “finger points” to the model poses no mathematical difficulty — we simply add more rows and columns to \bar{A} , z , and b . Finding the inverse or decomposition of 10x10 or even 100x100 matrices will not seriously tax a modern computer using standard linear algebra libraries, at least for non-realtime simulations. In this way we could produce a more realistic model of a finger. Instead of using two springs of equal strength, we could use multiple springs. The spring constants K and damping factors R of each point could be varied in order to better imitate the curved finger by using stronger springs for points at the centre of the finger and weaker springs for points at the edges of the finger. However, with the possible exception of the double bass playing jazz, plucking is not a large factor in normal instrument playing, and the current system creates plausible audio to casual listening. I therefore turn to the bowing algorithm.

2.1.3 Bow force on the string

When bowing the string, I only use two forces: F_0 represents the bow, while F_1 is a single finger force. F_1 remains modelled as a damped spring, while there are two ways of thinking about F_0 :

Adjusting modal values: our modal equations require a variable F_0 to update the string velocity v_0 in the desired manner. I represent this concept as $F_{\text{modal}}(\Delta v)$.

Force of friction: as the bow scrapes against the string, it creates a frictional force. I represent this concept as $F_{\text{friction}}(\Delta v)$.

Although $F_{\text{modal}}(\Delta v)$ and $F_{\text{friction}}(\Delta v)$ are the same force, it is useful to consider them separately for their derivation. This equality is crucial to avoiding a computationally expensive numerical solution to our differential equations. A graphical interpretation of this is shown in Figure 2.7: The equality is the intersection of the $F_{\text{friction}}(\Delta v)$ curve and the relevant $F_{\text{modal}}(\Delta v)$ diagonal line.

Both methods will benefit from defining the relative velocity between the bow and string,

$$\Delta v = v_0 - v_b \quad (2.32)$$

Deriving F_{modal}

Inserting y_1 and v_1 from (2.20) into (2.21) and solving for F_1 gives

$$\begin{aligned} F_1 &= -\frac{(B_{10}R_1 + A_{10}K_1)F_0 + R_1v_1^h + K_1y_1^h}{B_{11}R_1 + A_{11}K_1 + 1} \\ &= D_5F_0 + D_6v_1^h + D_7y_1^h \end{aligned} \quad (2.33)$$

Inserting (2.33) and (2.32) into v_0 from (2.20) and solving for F_0 gives

$$\begin{aligned} F_{\text{modal}} = F_0 &= \frac{(B_{11}R_1 + A_{11}K_1 + 1)(v_b + \Delta v - v_0^h) + (B_{01}K_1)y_1^h + (B_{01}R_1)v_1^h}{(B_{00}B_{11} - B_{01}B_{10})R_1 + (A_{11}B_{00} - A_{10}B_{01})K_1 + B_{00}} \\ &= D_1(v_b + \Delta v - v_0^h) + D_2y_1^h + D_3v_1^h \end{aligned} \quad (2.34)$$

In Appendix A, I conjecture that the denominator of (2.34) must be greater than zero.

Deriving F_{friction}

The “traditional” bow-string friction model is the hyperbolic friction curve with hysteresis (McIntyre et al. 1983), which expresses the frictional force in terms of F_b and Δv . The curve relies on the coefficients of static friction μ_s , dynamic friction μ_d , while μ_c indicates the slope².

$$F_{\text{traditional}} = \begin{cases} F_b \left(\mu_d + \frac{(\mu_s - \mu_d)\mu_c}{\mu_c - \Delta v} \right) & \text{if } \Delta v < 0 \\ -F_b \mu_s \leq F_{\text{traditional}} \leq F_b \mu_s & \text{if } \Delta v = 0 \\ -F_b \left(\mu_d + \frac{(\mu_s - \mu_d)\mu_c}{\mu_c + \Delta v} \right) & \text{if } \Delta v > 0 \end{cases} \quad (2.35)$$

²Notation: here there is another unfortunate clash of variable names. Some literature on bow friction uses v_0 to refer to the slope of the hyperbolic friction curve (μ_s in my notation). However, (Demoucron 2008) and (Demoucron & Rasamimanana 2009) use v_0^h to refer to the “hands-free” velocity of the string under the bow, which naturally suggests using v_0 for the string velocity at that point, as I have done. I find it much more natural to use μ to represent all friction parameters

$$\begin{aligned}
D_1 &= (B_{11}R_1 + A_{11}K_1 + 1)L_1 & D_2 &= B_{01}K_1L_1 \\
D_3 &= B_{01}R_1L_1 & D_4 &= \frac{1}{2D_1} \\
D_5 &= (B_{10}R_1 + A_{10}K_1)L_2 & D_6 &= R_1L_2 & D_7 &= K_1L_2 \\
L_1 &= \frac{1}{(B_{00}B_{11} - B_{01}B_{10})R_1 + (A_{11}B_{00} - A_{10}B_{01})K_1 + B_{00}} & L_2 &= \frac{-1}{B_{11}R_1 + A_{11}K_1 + 1}
\end{aligned}$$

Table 2.4: Coefficients for bow force on the string.

The friction curve is plotted in Figure 2.7; note that there is an “ambiguous” region where there is more than one possible intersection. At time t_a , the only intersection is at roughly $\Delta v = -1.25$; at time t_c the only intersection is $\Delta v = 0$. However, at time t_b , there are three candidates: $\Delta v \in \{-0.5, -0.1, 0\}$. This “Friedlander-Keller ambiguity”, named after the two mathematicians who first studied it, is resolved by adding a hysteresis rule to (2.35): if the system is currently sticking ($\Delta v = 0$), then it will continue to stick if possible; if the system is slipping ($\Delta v \neq 0$) then it will continue to slip if possible. Furthermore, the system will never reach the “middle value” between $\Delta v = 0$ and the farthest possible solution. In the case of Figure 2.7, if we progress from time $t_a \rightarrow t_b$ then $\Delta v = -0.5$; if we progress from time $t_c \rightarrow t_b$ then $\Delta v = -0.1$. Noise was added to the slipping state by multiplying the friction curve μ_e by a uniform random value $0.95 \leq u(t) \leq 1.0$ for every computation, defining $\mu_e = \mu_c u(t)$. This has the effect of adding pulsed noise at the slip/stick transitions, which can aid in establishing Helmholtz motion (Chafe 1990, Demoucron 2008).

$$F_{\text{friction}} = \begin{cases} F_b \left(\mu_d + \frac{(\mu_s - \mu_d)\mu_e}{\mu_e - \Delta v} \right) & \text{if } \Delta v < 0 \\ -F_b \mu_s \leq F_{\text{friction}} \leq F_b \mu_s & \text{if } \Delta v = 0 \\ -F_b \left(\mu_d + \frac{(\mu_s - \mu_d)\mu_e}{\mu_e + \Delta v} \right) & \text{if } \Delta v > 0 \end{cases} \quad (2.36)$$

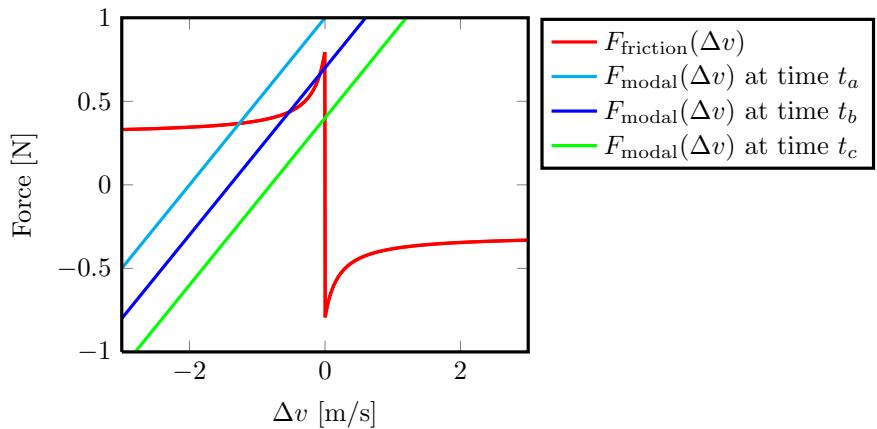


Figure 2.7: Hyperbolic friction curve using a common set of friction characteristics ($\mu_s = 0.8$, $\mu_d = 0.3$, $\mu_e = 0.2$) and bowing parameter $F_b = 1.0$. The randomness of μ_e alters the steepness of the curves in F_{friction} , but only by a small amount (the maximum and minimum μ_e curves are plotted but are indistinguishable at this resolution). The modal force comes from an open violin G string with bowing parameter $x_0 = 0.1L$, $v_b = 0.4$ and varying v_0^h .

Relative bow velocity

Calculating Δv depends on the current bow slip-state; given the hysteresis rule, we will consider each slipping state ($\Delta v < 0$, $\Delta v = 0$, $\Delta v > 0$) separately. If the current slip-state cannot be maintained, other states are tested as shown in Figure 2.8. Once Δv is derived, F_0 is calculated from (2.34).

Sticking: If the string is currently in the sticking state ($\Delta v = 0$), the bow continues to stick if

$$-F_b \mu_s \leq F_{\text{modal}} \leq F_b \mu_s \quad (2.37)$$

Slipping (negative): If the string is currently moving slower than the bow ($\Delta v < 0$), then we combine F_{modal} (2.34) with the relevant case of F_{friction} (2.36):

$$D_1(\Delta v + v_b - v_0^h) + D_2 y_1^h + D_3 v_1^h = F_{\text{modal}} = F_{\text{friction}} = F_b \left(\mu_d + \frac{(\mu_s - \mu_d)\mu_e}{\mu_e - \Delta v} \right) \quad (2.38)$$

Solving for Δv gives:

$$c_2 \Delta v^2 + c_1 \Delta v + c_0 = 0, \quad \text{with} \quad \begin{cases} c_2 = -D_1 \\ c_1 = -D_1(v_b - v_0^h - \mu_e) - D_2 y_1^h - D_3 v_1^h + F_b \mu_d \\ c_0 = \mu_e [D_1(v_b - v_0^h) + D_2 y_1^h + D_3 v_1^h - F_b \mu_s] \end{cases} \quad (2.39)$$

If the discriminant $\Delta = c_1^2 - 4c_0c_2$ is below 0, then there is no real solution for Δv so we reject this slip condition and check for a stick condition. If $\Delta = 0$ then there is only one solution. If $\Delta > 0$ then there are two solutions, but we always take the solution which is farthest from zero. Finally, recall that the formula for F_{friction} in (2.38) is only valid for $\Delta v < 0$, so we must reject any other solutions.

$$\Delta v = \min \left(\frac{-c_1 + \sqrt{c_1^2 - 4c_0c_2}}{2c_2}, \frac{-c_1 - \sqrt{c_1^2 - 4c_0c_2}}{2c_2} \right), \quad \text{provided that } \Delta v < 0 \quad (2.40)$$

Assuming that $D_1 > 0$ (see Appendix A), we can simplify (2.40) to

$$\Delta v = D_4 \left(c_1 - \sqrt{c_1^2 + 4c_0 D_1} \right), \quad \text{provided that } \Delta v < 0 \quad (2.41)$$

Slipping (positive): We repeat the same series of steps as with the negative slipping case.

$$D_1(\Delta v + v_b - v_0^h) + D_2 y_1^h + D_3 v_1^h = F_{\text{modal}} = F_{\text{friction}} = -F_b \left(\mu_d + \frac{(\mu_s - \mu_d)\mu_e}{\mu_e + \Delta v} \right) \quad (2.42)$$

Solving for Δv gives:

$$c_2 \Delta v^2 + c_1 \Delta v + c_0 = 0, \quad \text{with} \quad \begin{cases} c_2 = D_1 \\ c_1 = D_1(v_b - v_0^h + \mu_e) + D_2 y_1^h + D_3 v_1^h + F_b \mu_d \\ c_0 = \mu_e [D_1(v_b - v_0^h) + D_2 y_1^h + D_3 v_1^h + F_b \mu_s] \end{cases} \quad (2.43)$$

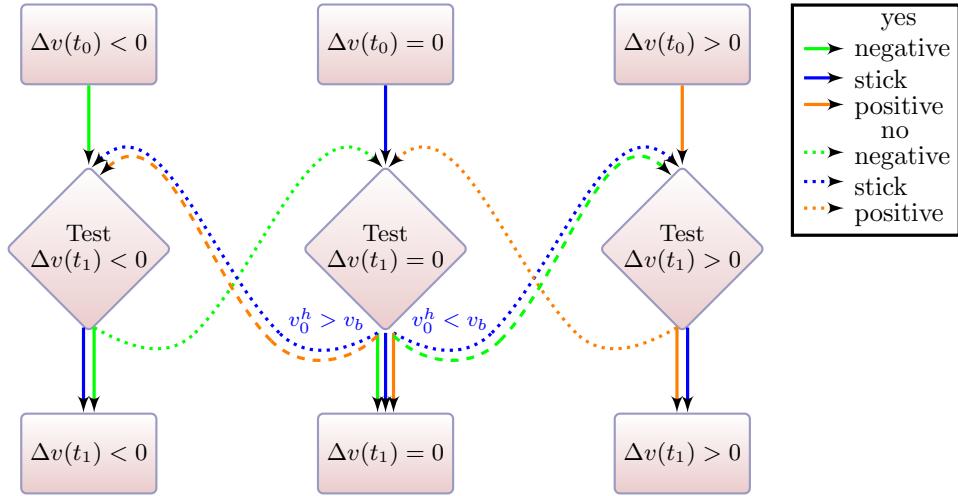


Figure 2.8: Bow slipping states.

If the discriminant $\Delta = c_1^2 - 4c_0c_2$ is below 0, reject this slip condition, otherwise

$$\Delta v = \max \left(\frac{-c_1 + \sqrt{c_1^2 - 4c_0c_2}}{2c_2}, \frac{-c_1 - \sqrt{c_1^2 - 4c_0c_2}}{2c_2} \right), \text{ provided that } \Delta v > 0 \quad (2.44)$$

With the assumption that $D_1 > 0$, this produces

$$\Delta v = D_4 \left(-c_1 + \sqrt{c_1^2 - 4c_0D_1} \right), \quad \text{provided that } \Delta v > 0 \quad (2.45)$$

Transition between slip states

If the bow is sticking to the string but it fails the test in (2.37), the bow begins to slip. To determine whether $\Delta v > 0$ or $\Delta v < 0$, we examine the relative velocity of the string under the bow if there were no external forces. If the string would be moving slower than the bow if there were no external forces ($v_0^h < v_b$), then the effect of the bow will cause the string to move faster ($F_0 > 0$). This occurs when $\Delta v < 0$, so we move to that state. A similar argument is made for $v_0^h > v_b$ implying that $\Delta v > 0$.

Most friction models (including (Demoucron 2008)) assume that the string will never be moving faster than the bow. This is a fair assumption when bowing strings with relatively low fundamental frequencies and the string always begins from rest. However, when bowing the violin A or E strings, the string velocity at x_0 can exceed the bow velocity. Ideally there would be at least one sample of stick-state before reaching the higher velocity, but this does not always occur. If the slip-state is not allowed to “jump” directly from negative slipping to positive slipping, the system can become unstable. This is discussed in Section 2.3.3. For that reason, Figure 2.8 allows the system to begin in the negative slipping state but end in the positive slipping state (and vice versa). Allowing the string to jump from negative to positive slip-states is a useful “safety valve” which avoids the instability without significantly increasing the amount of computations due to a higher sampling rate.

Final forces

After determining the slip-state and Δv , F_0 is calculated from (2.34) and F_1 is calculated from (2.33).

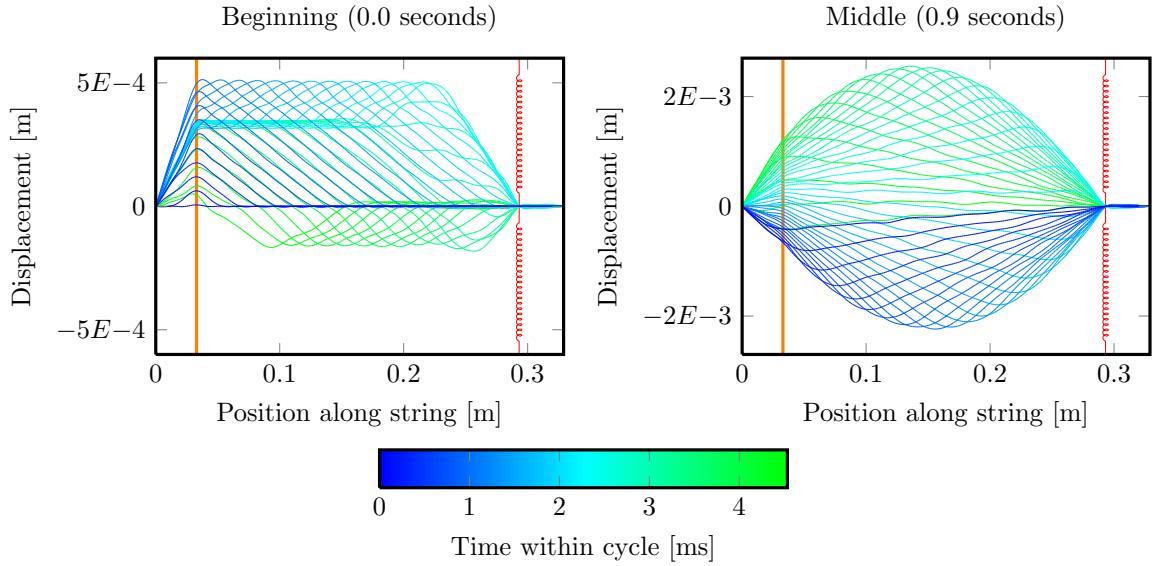


Figure 2.9: Movement of a bowed string, downsampled by a factor of 5. Finger position $x_f = 0.891L$ (220 Hz, expected cycle 4.5 ms), bowing parameters $x_b = 0.1L$, $F_b = 1.0$, $v_b = 0.5$. The orange line represents the bow, while the red spring represents the finger as usual.

Audio 2.2: Normal bowed fingered string.

<http://percival-music.ca/dissertation/a.2.2.bow-finger-movie-middle-violin-g.wav>

Video 2.3: Normal bowed fingered string, beginning.

<http://percival-music.ca/dissertation/v.2.3.bow-finger-movie-begin-violin-g.mpeg>

Video 2.4: Normal bowed fingered string, middle.

<http://percival-music.ca/dissertation/v.2.4.bow-finger-movie-middle-violin-g.mpeg>

In the videos, a solid line for the bow indicates a stick-state, while a light line indicates a slip-state.

Accelerating the bow

In order to facilitate the establishment of Helmholtz motion when bowing a string from rest, as noted by (Guettler 2002), the model accepts an additional input parameter: bow acceleration a_b . If this parameter is set, then v_b is updated at each time step,

$$v_b(t_{i+1}) = v_b(t_i) + a_b dt \quad (2.46)$$

until v_b reaches a target velocity v_b^t .

Bowed string movement

Figure 2.9 shows a typical fingered bow-stroke. An analysis of professional violinists' bowing (Guettler 2002) showed that fewer than 50% of notes establish perfect Helmholtz motion from the outset; this is true of the “beginning” plot in which the bow’s motion does not end at the same position in which it began. By contrast, the “middle” plot shows that the string’s motion completes one full cycle in 4.5 ms. At the note beginning, the string’s movement is highly biased in the direction of the bow’s movement, whereas after the string has “settled down” into normal Helmholtz motion, the string’s movement is almost (but not quite) evenly split between positive and negative y directions. The interplay of bow slip-states and string motion during Helmholtz motion is quite visible in Video 2.4.

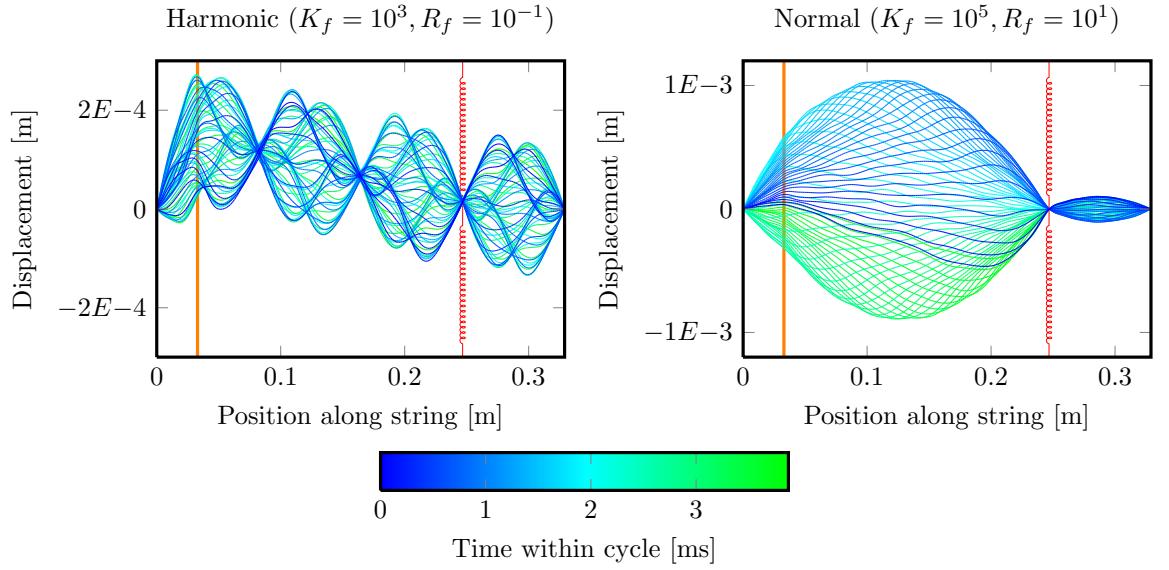


Figure 2.10: Bow harmonics, downsampled by a factor of 3. Left: natural harmonic (very light left-hand finger), right: normal bowing (normal left-hand finger). Both simulations used the same finger position $x_f = 0.75L$ (expected frequency 261 Hz, cycle 3.8 ms) and bowing parameters of $x_b = 0.1L$, $F_b = 0.5$, $v_b = 0.3$ on the violin G string. Both plots begin at 1.0 seconds. The large vibrations above the finger position are expected in the harmonic note, but not desired for the normal note.

Audio 2.3: Natural harmonic.

<http://percival-music.ca/dissertation/a.2.3.bow-harmonic-light-violin-g.wav>

Video 2.5: Natural harmonic

<http://percival-music.ca/dissertation/v.2.5.bow-harmonic-light-violin-g.mpeg>

Audio 2.4: Normal Helmholtz vibrations.

<http://percival-music.ca/dissertation/a.2.4.bow-harmonic-normal-violin-g.wav>

Video 2.6: Normal Helmholtz vibrations.

<http://percival-music.ca/dissertation/v.2.6.bow-harmonic-normal-violin-g.mpeg>

Natural harmonics

Gently touching a vibrating string at nodes can produce a “flute-like” sound, known as “natural harmonics” or often simply “harmonics”³. The physical model accommodates harmonics, as shown in Figure 2.10. Plucked harmonics can also be created, but it is quite difficult to find the right balance between left-hand finger strength and the string decay. Since the bow provides a constant input of energy, bowed harmonics are much easier to produce in both real life and the model.

³“Artificial harmonics” also exist, but they are an advanced violin technique and are not supported in this model due to requiring two left-hand fingers on the string

2.1.4 String states

Depending on the type of external actions, the string simulation will vary between 5 states, as shown in Figure 2.11.

Bow: If there is a bow on the string, then the equations in Section 2.1.3 are used. When the bow is removed from the string, the state changes to a release-state.

Pluck: If a pluck has begun, the equations in the first part of Section 2.1.2 are used. After T_p seconds, the string is automatically released and changes state.

Release (finger): If a left-hand finger is on a string, the equations in the second part of Section 2.1.2 are used. When the string's vibrations are insignificant, the string calculations are turned off.

Release (no finger) : If there is no left-hand finger, then there are no external forces and the string vibrates freely; only equations in Section 2.1.1 are used. In particular, $a_n \leftarrow a_n^h$ and $\dot{a}_n \leftarrow \dot{a}_n^h$. When the string's vibrations are insignificant, the string calculations are turned off.

Off : The string does not move at all; $A[t] \leftarrow 0$ and $H[t] \leftarrow 0$.

Turning off the string

If the string is not being bowed or plucked, the vibrations continually lessen and after some time they will be too small to provide any output in the 16-bit-quantized signal. For each complete buffer of each string, we check the sum of squares

$$A_{\text{mag}} = \sum_i A[i]^2 \quad (2.47)$$

against a threshold value A_{\min} defined in Section 3.5.5. If the value is smaller, then we disable computation for this string and set $a_n = 0$, $\dot{a}_n = 0$. This check is only performed on a full buffer of samples; if a smaller buffer size was requested, the check is omitted to avoid premature silencing.

External applications may wish to add dither to the final output. In some cases, adding small amounts of randomness (“dither”) to the output signal can improve the human perception of the sound. I decided that dither should not be part of the physical simulation itself, since the amount and shaping may depend on the specific use case of the simulation.

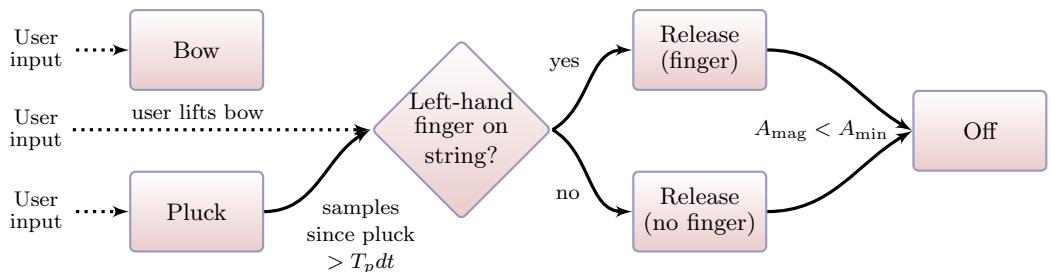


Figure 2.11: String states. Solid lines indicate automatic progress; dotted lines indicate user input. Whenever there is user input, the string state jumps back to the left-hand side.

2.1.5 Instrument body simulation

The instrument body and bow are modelled as linear time-invariant systems. Each string is simulated for n samples, then the results are combined according to the sample rates as shown in Figure 2.2. Finally, the signals are quantized to 16-bit integers for convenient use with audio software.

The output of a linear time-invariant system is found by convolving the input signal with the impulse response of the system. Convolution can be considered to be a finite impulse response (FIR) filter, and FIR filters with a large number of coefficients are almost always most efficiently performed with FFTs via the overlap-add method (Stockham 1966). Given a block of an input signal $x_b[n]$ and a convolution kernel $h[n]$, the convolution can be performed as

$$y_b[n] = \text{FFT}^{-1}(\text{FFT}(x_b[n]) \circ \text{FFT}(h[n])) \quad (2.48)$$

However, since the FFT will be longer than the block size of the input signal, there is important data “overlapping” into the next x_b block. This “overlapping” data is retained and added to the next x_b . Given a kernel length of M and block size of L , the FFT length N must be chosen such that $N \geq L + M - 1$. Note that $\text{FFT}(h[n])$ does not change throughout the simulation, so it need only be calculated once.

Multi-sample rate strings

All instruments in this simulation output an audio signal at a sample rate of 22050 Hz, but this is below the Nyquist rate for some the upper modes of some strings. Furthermore, as is discussed in Section 2.3, the sample rate can significantly change the results of bowing a string. To mitigate these problems, many strings are computed at a frequency which is an integer multiple of 22050 Hz. Each string is computed at a frequency multiplier M_f (ranging from 1x to 4x), with the exact multipliers listed in Section 3.5.2.

When combining string signals, I first add signals of the same sample rate, then apply the filters discussed below, then decimate the resulting signal by the relevant frequency multiplier. The resulting signal(s) are summed, forming the final output. The entire process involves two filters: A low-pass anti-alias filter to permit the downsampling, and the FIR body filter. Since both filters are linear time-invariant, I combine them to form a single filter.

In Section 3.3.2 I recorded 4096 samples of violin, viola, and cello impulse responses at 22050 Hz. That impulse response is upsampled to match the desired frequency multiplier. The overall instrument calculates at most 512 samples at a time. After examining the spectrums of instrument body responses with 511 and 1535 samples, I decided to use the longer impulses to retain spectral resolution. This means that with a frequency multiplier $M_f = 1$, I must use an FFT of length $N = 2048$; for $M_f = 4$, I must use an FFT of length $N = 8192$. For $M_f > 1$, the body response is upsampled accordingly. The haptic signals use an FIR filter of order 512, resulting in FFTs of length $N = 1024M_f$.

After applying the FIR filter, the signals are decimated. $A[t]$ is decimated by M_f , while $H[t]$ is decimated by $2M_f$.

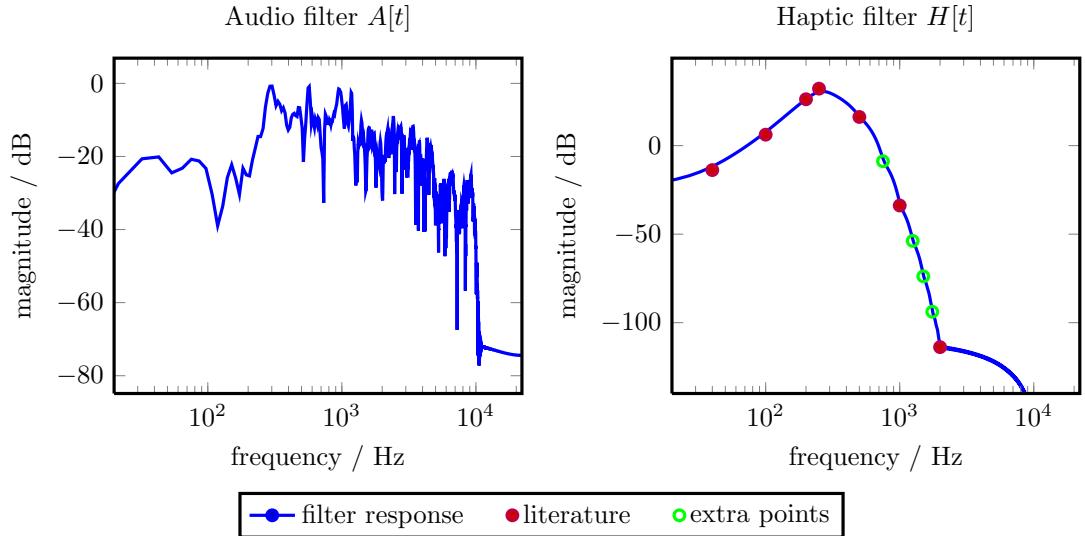


Figure 2.12: Frequency response of violin instrument body filters. Left: An example of the combined response of the violin impulse response with the low-pass filter. Right: An example of the tactile sensitivity filter designed with the window method; a few extra points were added to make the designed filter have a smoother shape in the region between 250 Hz and 2000 Hz. The overall gains of the two filters were chosen such that there would be a clear signal after quantizing to 16-bit integers.

Kernels

The convolution kernel for audio output comes from the impulse responses measured in Section 3.3.2. The impulse response is upsampled by the relevant frequency multiplier, then filtered with a low-pass FIR filter with a cut-off of 10000 Hz designed with the window method using a Hamming window of order 101 (Smith 2011). An example is shown in Figure 2.12. To avoid the upsampled impulse responses resulting in louder strings than the non-upsampled strings, each impulse response is normalized by its (post-upsampling) length.

The convolution kernel for the haptic output was generated from the literature on human tactile finger sensitivity. Data points from two papers were combined and used to generate an FIR filter using the window method with a Hamming window of order 512. The frequency response of the resulting filter is shown in Figure 2.12. In particular, (Bolanowski et al. 1988) provided data from 0.4 Hz to 500 Hz, while (Wyse et al. 2012) provided data from 250 Hz to 2000 Hz. Since the sensitivity was already lower than -100 dB at 6000 Hz, no additional anti-alias filter was applied.

It must be acknowledged that this is only a very rough estimate of tactile sensitivity and that the two papers used different measurement methods; combining data points in this manner is not scientifically valid. However, it provides a plausible ‘‘first estimate’’, as the actual finger sensitivity is unlikely to differ by an order of magnitude, and this filter can easily be changed later when more data of tactile sensitivity emerges.

2.2 Design decisions and consequences for finger actions

This section examines the physical model in greater detail, and discusses various design decisions. Unless otherwise specified, all frequency plots were generated from an FFT with a Blackman-Harris window on $2^{16} = 65536$ samples of audio (≈ 1.5 seconds at 44100 Hz).

I encountered three main issues when designing the left-hand finger and right-hand plucking models: The need for damped springs, the need for at least two forces for right-hand plucking, and the need for at least two forces for the left-hand finger during string release.

These difficulties were most prominent when the finger or pluck positions were at “strongly rational” positions such as $\frac{1}{2}L$, $\frac{2}{3}L$, or $\frac{3}{4}L$. The latter two correspond to the normal left-hand finger positions of the 3rd and 4th fingers, while programmers testing the physical model are quite likely to select values such as $0.5L$ or $0.25L$ for the plucking position, so it is important to ensure that the model can handle such values.

2.2.1 Damped spring action

I chose to model the left-hand finger and right-hand plucking finger as damped springs, rather than either undamped springs or viscous dampers alone. The effects of damping the string is shown in Figure 2.13, while the exact value of the constants are selected in Section 3.5.3.

A viscous damper alone can function well for a single note (be it plucked or bowed). This method is used in (Demoucron & Rasamimanana 2009), although that paper only included bowed notes and not plucked notes. However, consider the case of plucking an open string and then placing a left-hand finger on the string a short time after plucking (say, $x_f = 0.25L$ and $t = 0.5$ seconds). The open

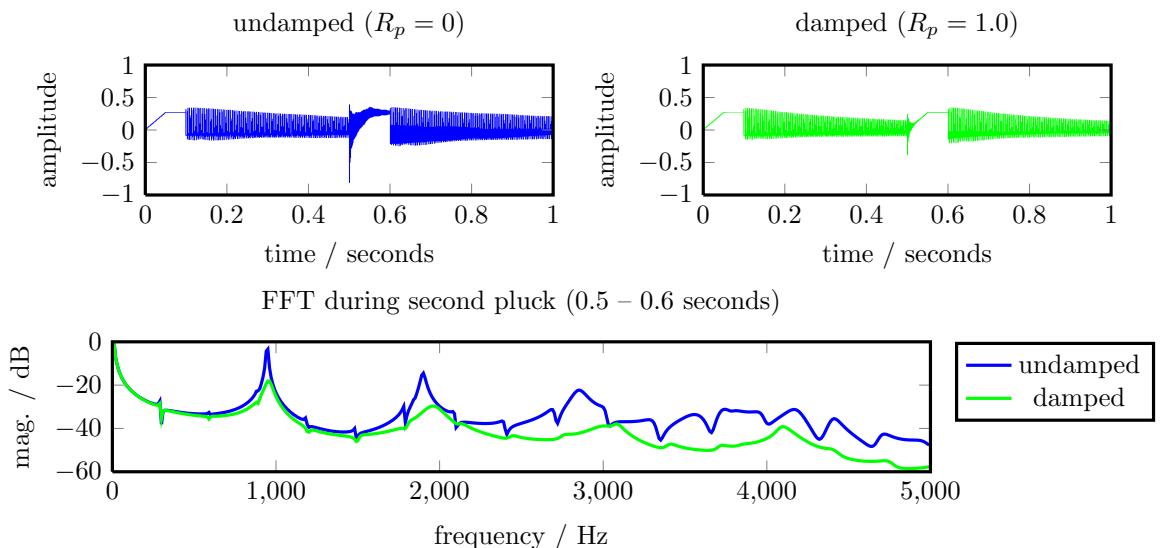


Figure 2.13: Two plucks with damped and undamped plucking springs. Violin G string, first finger $x_f = 0.891L$, two plucks $x_p = 0.2L$. The undamped spring results in highly undesirable behaviour when the second pluck begins at $t = 0.5$ seconds. Note that the damped pluck has a flatter frequency response during the plucking portion (as is desired in this case).

Audio 2.5: Two plucks, undamped

<http://percival-music.ca/dissertation/a.2.5.pluck-finger-two-plucks-undamped-violin-g.wav>

Audio 2.6: Two plucks, damped

<http://percival-music.ca/dissertation/a.2.6.pluck-finger-two-plucks-damped-violin-g.wav>

string will be vibrating, so unless the string's transverse displacement at $x = 0.25L$ happens to be 0, the effect of the finger will be to "fix" the string's displacement at that point to be non-zero. This is not desirable; a violinist's left-hand finger usually fixes the string's transverse displacement at the point of contact to be close to zero.

An undamped spring alone can also function well for a single note; this method is used in (Demoucron 2008). However, there are two situations where this is not suitable. First, a real string decays faster when a finger is pressed against the string compared to plucking an open string, since the rounded finger and flesh dampen the vibrations. This additional damping is shown experimentally in Section 3.5.3. The decay could be imitated by changing the r_n damping factors in the string constants whenever a finger is pressed against the string, but it is much clearer to add the damping via damped springs. Failing to include such additional damping results in all notes sounding like open strings. This is not desirable, but it is a relatively minor problem compared to other problems discussed in this section. Second, the damping is very useful to deaden vibrations when plucking an already-moving string. When using an undamped spring to model the right-hand pluck, attempting to pluck an already-moving spring causes a loud "wooden tick" sound. By contrast, modelling the pluck with a damped spring produces an acceptable sound.

2.2.2 Two forces for right-hand plucking

In many cases, using a single force to model the right-hand plucking force is sufficient. However, consider plucking an open string at $x_p = 0.5L$, as shown in Figure 2.14. There are no left-hand finger forces, so the only external actions on the string come from the pluck. If the pluck is modelled with a single (infinitely thin) point, then none of the even modes will gain any energy as they all have a node at $x = 0.5L$. For this reason, it is useful to model the pluck with at least two forces. Even if x_p is a node for many modes of vibration, $x_2 = x_p + W_p$ will only be a node if $\frac{L}{W_p}$ is an exact multiple of the mode number. For example, if a string length is 0.33 m and we use $N = 33$ modes, $W_p = 0.01$ m, and x_p is a node of mode 33, then $x_p + W_p$ will also be a node. However, such concerns only apply to relatively higher-order modes, which will only result in problems in the upper spectrum.

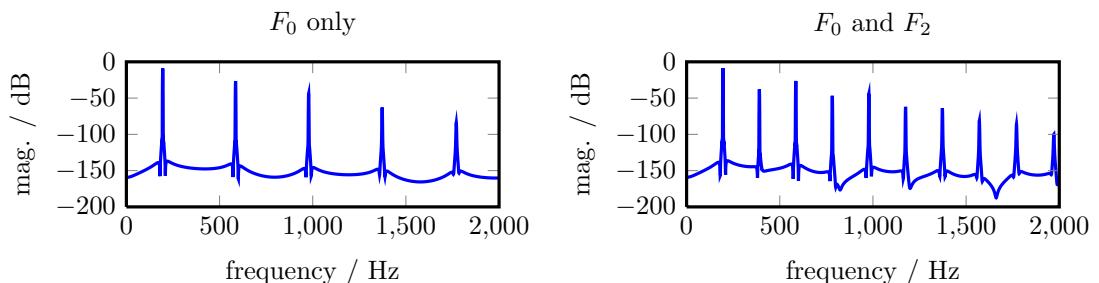


Figure 2.14: Comparison of one and two plucking forces. Violin G string, open string $x_f = 0.0$ (expected frequency 196 Hz), pluck $x_p = 0.5L$. Left: one plucking force. Right: two plucking forces. Note that with one plucking force, all the even partials above 196 Hz are missing; musically we desire a full set of partials as shown in the right-hand plot.

Audio 2.7: Plucking an open string with one pluck force.

<http://percival-music.ca/dissertation/a.2.7.pluck-open-one-pluck-force-violin-g.wav>

Audio 2.8: Plucking an open string with two pluck force.

<http://percival-music.ca/dissertation/a.2.8.pluck-open-two-pluck-force-violin-g.wav>

2.2.3 Two forces for the left-hand finger

A single finger force causes strong beating when the finger is placed at “strongly rational” positions, as shown in Figure 2.15. I will examine why the single finger force is not sufficient in greater detail.

In order to eliminate as many variables as possible, I define an ideal *unit string* in Table 2.5, which reduces the string model to the bare minimum by eliminating string stiffness and modal decay. The modal frequencies are thus exact multiples with no variation due to stiffness or damping frequencies. However, even this simplified model produces clearly visible beating as seen in Figure 2.16. This unwanted frequency comes from F_1 , so we investigate the actions of the finger force in greater detail.

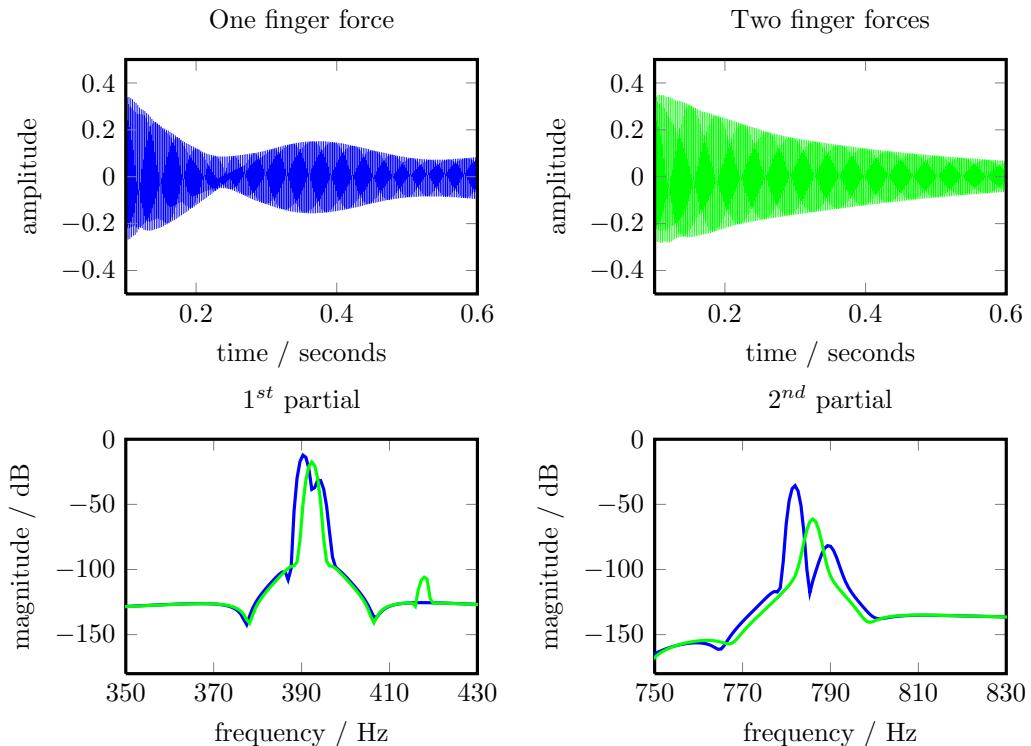


Figure 2.15: Comparison of one and two finger forces during plucking. Violin G string, finger $x_f = 0.5L$ (expected frequency 392 Hz), pluck $x_p = 0.2L$. Left: modelling the finger with a single force during string release. Right: modelling the finger with two forces during string release. Bottom: close-up of the first and second partials. The “split peaks” audio from the single finger force version sounds like two strings plucked simultaneously. The unwanted “extra” peak at 420 Hz in the two finger force version is an oddity, but it is preferable to the full “split peaks” of the single-force version.

Audio 2.9: Pluck with one finger force.

<http://percival-music.ca/dissertation/a.2.9.pluck-finger-force-one-violin-g.wav>

Audio 2.10: Pluck with two finger forces.

<http://percival-music.ca/dissertation/a.2.10.pluck-finger-force-two-violin-g.wav>

$$T = 400 \quad L = 2.0 \quad \rho_L = 10^{-4} \quad d = 0 \quad E = 0 \quad N = 3 \quad r_n = [0, 0, 0]$$

$$dt = \frac{1}{44100} \quad w_n = [500 \cdot 2\pi, \quad 1000 \cdot 2\pi, \quad 1500 \cdot 2\pi]$$

One finger force modelled as an infinitely strong undamped spring ($K_p = \infty$, $R_p = 0$).

Table 2.5: Unit string with simple physical constants: It is perfectly elastic with no modal decay.

Since the unit string has only three modes and one finger force at $x_1 = 0.5L$, the eigenvectors are $\phi_n(x_1) = [1, 0, -1]$. Mode 2 thus has no effect on F_1 , and in turn F_1 has no effect on mode 2. We can therefore consider the “string release” modelling as two completely separate systems: Mode 2 by itself, and the combination of modes 1 and 3. The “expected” lower peak in Figure 2.16 comes from mode 2, so we omit that from future consideration.

To examine the behaviour of modes 1 and 3, I turn to modern control theory. Given the state-space representation⁴ of a discrete time-invariant system where $\chi(t)$ is the state of the system, $u(t)$ is the control vector, Φ is the state transition matrix, Γ is the input matrix, and H is the output matrix,

$$\begin{aligned}\chi[k+1] &= \Phi\chi[k] + \Gamma u[k] \\ y[k] &= H\chi[k]\end{aligned}\tag{2.49}$$

The general solution for the transfer function (Warwick 1989), after taking the Z-transform, is

$$G(z) = \frac{H \cdot \text{adjoint}(zI - \Phi) \cdot \Gamma}{\det(zI - \Phi)}\tag{2.50}$$

⁴Notation: The literature generally writes (2.49) as $\boldsymbol{x}[k+1] = A\boldsymbol{x}[t] + B\boldsymbol{u}[k]$; $y[t] = C\boldsymbol{x}[k] + D\boldsymbol{u}[k]$. However, I have already used x , A , B , and D , so we turn to less commonly-used symbols for the state-space representation. In addition, there is no feed-forward term $D\boldsymbol{u}[k]$ in the model so I omit it.

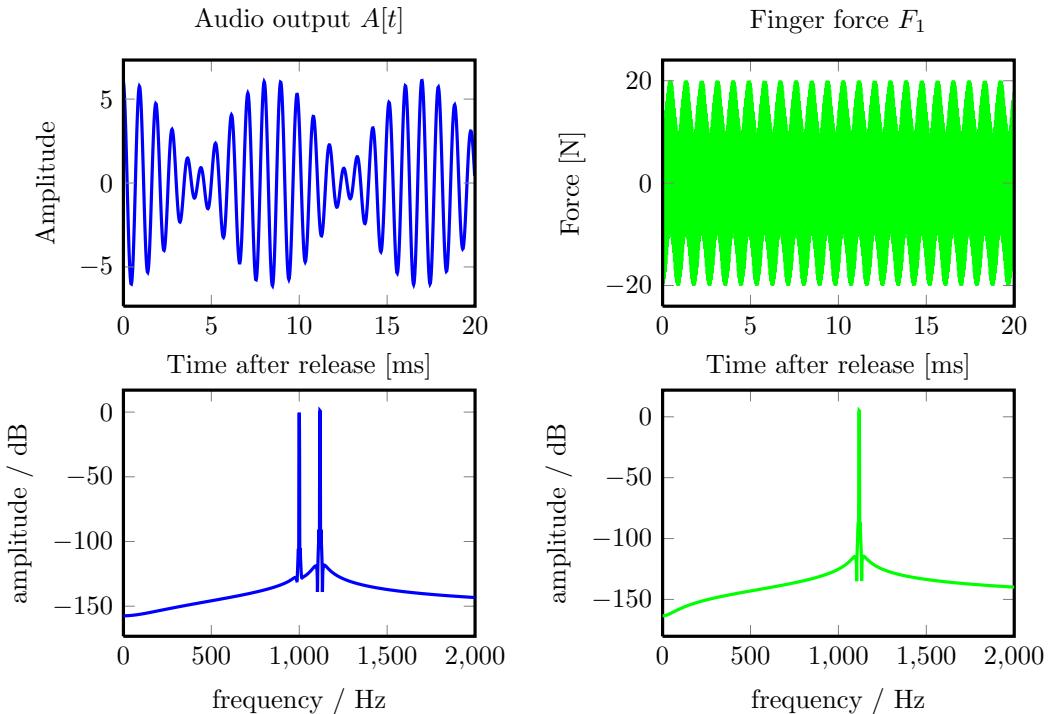


Figure 2.16: Beating in the unit string. The finger is at x_f (expected frequency 1000 Hz). The lower peak matches this expectation, while the upper peak is an unexpected 1118 Hz. In addition, the sidebands appear to come from this upper peak, with the lower peak appears to be “tacked on” to the plot. As may be deduced from the time-domain plot for F_1 , there is a great deal of energy at the Nyquist frequency (not shown in the “zoomed-in” frequency plot).

Audio 2.11: Beating in the plucked unit string.

<http://percival-music.ca/dissertation/a.2.11.pluck-beating-unit.wav>

In order to derive Φ , I will rewrite the full set of equations for the physical model to apply only to the unit string (i.e. only three modes and one infinitely strong finger force at $x_1 = 0.5L$). To avoid confusion between the real equations for the physical model and these highly-simplified equations, I will shade the background with a faint gray colour and omit equation numbers.

For modes 1 and 3 of the unit string, the equations describing the physical model are

$$\begin{aligned}
 a_1^h(t_1) &= X_{11}a_1(t_0) + X_{21}\dot{a}_1(t_0) & a_3^h(t_1) &= X_{13}a_3(t_0) + X_{23}\dot{a}_3(t_0) \\
 \dot{a}_1^h(t_1) &= Y_{11}a_1(t_0) + Y_{21}\dot{a}_1(t_0) & \dot{a}_3^h(t_1) &= Y_{13}a_3(t_0) + Y_{23}\dot{a}_3(t_0) \\
 y_1^h &= \sum_{n=1}^N \phi_n(x_1)a_n^h = a_1^h - a_3^h & A_{11} &= \sum_{n=1}^N \phi_n(x_1)^2 X_{3n} = X_{31} + X_{33} \\
 \check{f}_1(t_1) &= \phi_1(x_1)F_1(t_1) = F_1(t_1) & \check{f}_3(t_1) &= \phi_1(x_1)F_1(t_1) = -F_1(t_1) \\
 a_1(t_1) &= a_1^h + X_{31}\check{f}_1 & a_3(t_3) &= a_3^h + X_{33}\check{f}_3 \\
 \dot{a}_1(t_1) &= \dot{a}_1^h + Y_{31}\check{f}_1 & \dot{a}_3(t_3) &= \dot{a}_3^h + Y_{33}\check{f}_3 \\
 F_1(t_1) &= \frac{-y_1^h}{A_{11}}
 \end{aligned}$$

Combining all those equations allows us to describe the system as:

$$\begin{aligned}
 a_1(t_1) &= \frac{X_{11}X_{33}}{X_{33}+X_{31}}a_1(t_0) & + \frac{X_{13}X_{31}}{X_{33}+X_{31}}a_3(t_0) \\
 &+ \frac{X_{21}X_{33}}{X_{33}+X_{31}}\dot{a}_1(t_0) & + \frac{X_{23}X_{31}}{X_{33}+X_{31}}\dot{a}_3(t_0) \\
 a_3(t_1) &= \frac{X_{11}X_{33}}{X_{33}+X_{31}}a_1(t_0) & + \frac{X_{13}X_{31}}{X_{33}+X_{31}}a_3(t_0) \\
 &+ \frac{X_{21}X_{33}}{X_{33}+X_{31}}\dot{a}_1(t_0) & + \frac{X_{23}X_{31}}{X_{33}+X_{31}}\dot{a}_3(t_0) \\
 \dot{a}_1(t_1) &= \frac{X_{11}X_{33}+X_{31}Y_{11}-X_{11}Y_{31}}{X_{33}+X_{31}}a_1(t_0) & + \frac{X_{13}X_{31}}{X_{33}+X_{31}}a_3(t_0) \\
 &+ \frac{X_{33}Y_{21}+X_{31}Y_{21}-X_{21}Y_{31}}{X_{33}+X_{31}}\dot{a}_1(t_0) & + \frac{X_{23}X_{31}}{X_{33}+X_{31}}\dot{a}_3(t_0) \\
 \dot{a}_3(t_1) &= \frac{X_{11}Y_{33}}{X_{33}+X_{31}}a_1(t_0) & + \frac{X_{33}Y_{13}+X_{31}Y_{13}-X_{13}Y_{33}}{X_{33}+X_{31}}a_3(t_0) \\
 &+ \frac{X_{21}Y_{33}}{X_{33}+X_{31}}\dot{a}_1(t_0) & + \frac{X_{33}Y_{23}+X_{31}Y_{23}-X_{23}Y_{33}}{X_{33}+X_{31}}\dot{a}_3(t_0)
 \end{aligned}$$

As a quick sanity check, note that $a_1(t_1) = a_3(t_1)$, which matches the empirical modal displacements seen in Figure 2.17.

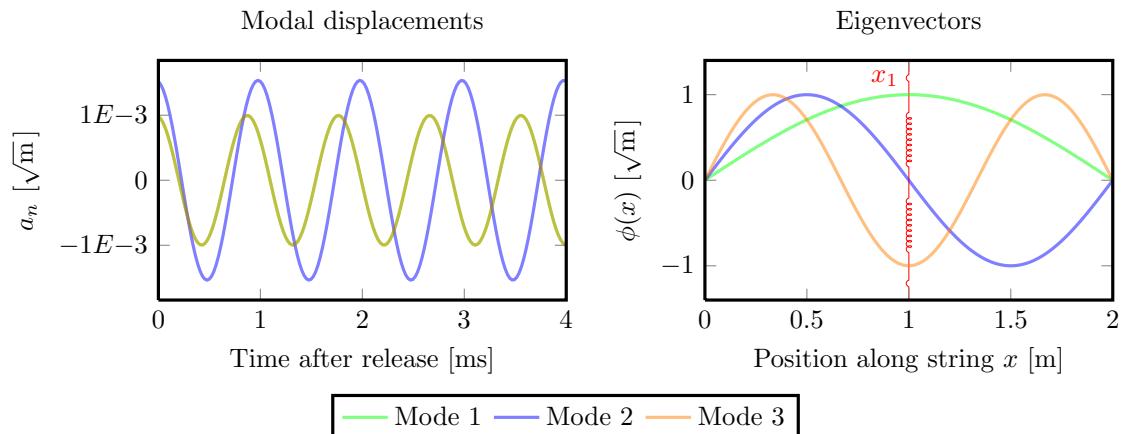


Figure 2.17: Modal displacements during beating in the unit string, $x_1 = 0.5$. In the left plot, modes 1 and 3 have the same values.

Intuitively, since $\phi_1(x_1) = 1$ and $\phi_3(x_1) = -1$, modes 1 and 3 must have equal displacement in order for $y(x_1, t) = 0$. With this intuition, we can simplify the system from 4 equations to 3 equations by substituting $a_1(t_1) = a_3(t_1)$. To describe the system as state equations, we define the state variables as

$$\chi = \begin{bmatrix} a_1(t) \\ \dot{a}_1(t) \\ \dot{a}_3(t) \end{bmatrix}$$

The state transition matrix is then

$$\Phi = \begin{bmatrix} \frac{X_{11}X_{33} + X_{13}X_{31}}{X_{33} + X_{31}} & \frac{X_{21}X_{33}}{X_{33} + X_{31}} & \frac{X_{23}X_{31}}{X_{33} + X_{31}} \\ \frac{X_{13}Y_{31} - X_{11}Y_{31} + X_{33}Y_{11} + X_{31}Y_{11}}{X_{33} + X_{31}} & \frac{X_{33}Y_{21} + X_{31}Y_{21} - X_{21}Y_{31}}{X_{33} + X_{31}} & \frac{X_{23}Y_{31}}{X_{33} + X_{31}} \\ \frac{X_{11}Y_{33} - X_{13}Y_{33} + X_{33}Y_{13} + X_{31}Y_{13}}{X_{33} + X_{31}} & \frac{X_{21}Y_{33}}{X_{33} + X_{31}} & \frac{X_{33}Y_{23} + X_{31}Y_{23} - X_{23}Y_{33}}{X_{33} + X_{31}} \end{bmatrix}$$

Substituting our constants, this gives

$$\Phi = \begin{bmatrix} 0.98734 & 1.130911 E-5 & 1.12708 E-5 \\ -1116.164 & -0.0029584 & 0.99366 \\ -1112.383 & 0.99366 & -0.0097019 \end{bmatrix}$$

The poles (i.e. the eigenvalues of Φ) are

$$\lambda = 0.98734 - 0.15862j, \quad 0.98734 + 0.15862j, \quad -1.0$$

The zeros depend on Γ and H . The output of the model (i.e. bridge force in (2.17)) is a linear combination of modal displacements. Since $a_1 = a_3$ and I am only concerned with the frequency and not the overall scaling, I define

$$H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

Γ is more difficult to determine, as it involves the plucking force F_0 which we omitted in the previous discussion as it plays no role in modelling the string release for the unit string. For simplicity, I assume that external force will change the modal displacements a_1 and a_3 but will not alter their velocities. Therefore

$$\Gamma = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

This gives the zeros as

$$z^2 + 0.0166z - 0.9834 = (z - 0.98734)(z + 1)$$

Combining the poles and zeros gives the transfer function

$$G[z] = \frac{(z - 0.98734)(z + 1)}{(z - 0.98734 - 0.15862j)(z - 0.98734 + 0.15862j)(z + 1)}$$

A set of typical signal processing plots of this system is shown in Figure 2.18. The important thing to note is the pole at frequency $\arctan(\frac{0.15862}{0.98734}) = 0.159293$ rad/sec, or 1118 Hz in our system at 44100 Hz — exactly where the unwanted frequency was found in the simulation.

This analysis of the behaviour of the simpler “unit string” is useful reassurance that the unwanted 1118 Hz peak seen in Figure 2.16 is not the result of programming error or floating-point rounding errors. I have mathematically shown that in the unit string, F_1 acts to produce a peak at 1118 Hz, while the second mode (unaffected by F_1) will vibrate at the desired 1000 Hz.

A full analysis of the normal (non-unit) string would be much more complicated. In order to deal with an arbitrary finger position, it would not be possible to decompose the system into multiple separate sub-systems as was done with the unit system (i.e. considering mode 2 separately from modes 1 and 3). In order to describe the normal system, we would need a dense matrix of $2N \times 2N$ (i.e. 80×80 for our $N = 40$ system). However, such a system would be of limited value, as it would only be an accurate simulation of plucks. The bow-string friction force calculations involve a square root and a random value, which would require non-linear robust stochastic control methods to analyze.

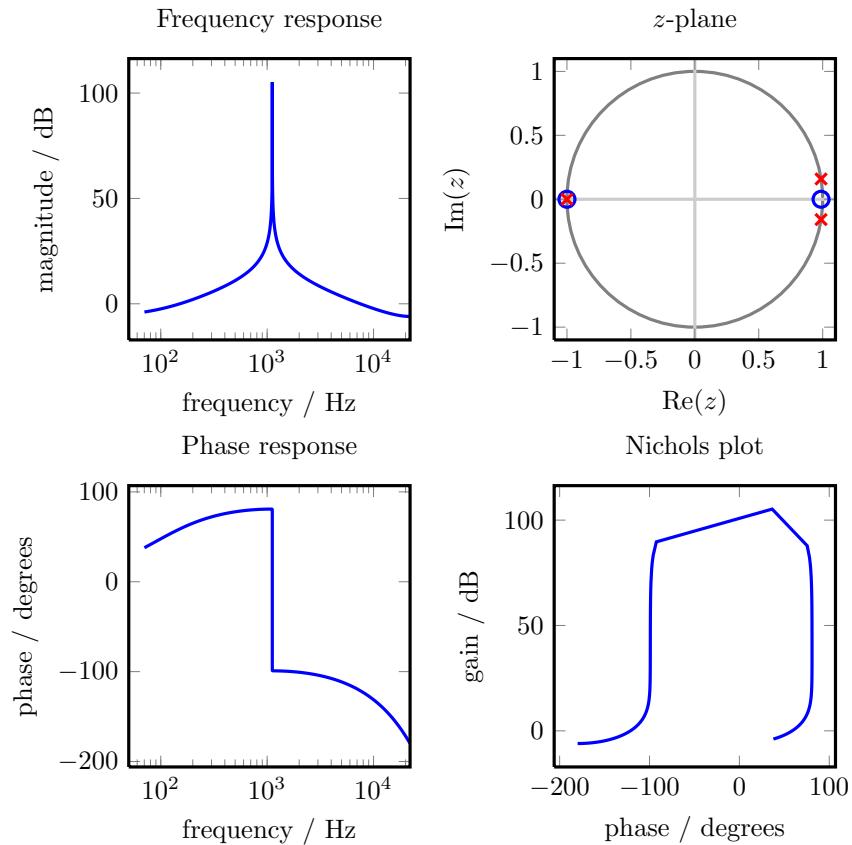


Figure 2.18: System behaviour of modes 1 and 3 in the unit string.

2.2.4 Three finger forces in total

Having seen the need for at least two forces for the pluck and at least two forces for the left-hand finger during string release, the next questions are whether two forces for each is enough. As discussed in Figure 2.1.2, the model uses a total of three forces: During plucking, there are three forces (two for the pluck, one for the finger); during release, there are two active forces (for the finger).

The number of forces during the pluck is investigated in Figure 2.19 (left): I compare the model's usual three forces during the pluck against four forces (two for the pluck, two for the finger). The two audio files are almost identical (judged by mixing the first file with the inverted second file), as can be seen from the extremely close match of spectrums.

The final question is whether we need to use all three finger forces during string release, or whether only need two (and can then leave one force as 0). This is tested in Figure 2.19 (right): I compare the model's usual three forces (but only two during pluck release) with three forces throughout (with $x_2 = x_f + \frac{W_f}{2}$ during release). More power is concentrated in lower spectral peaks, but this difference is not audible with casual listening. Since there is no audible difference and three forces require more calculations, I use two forces for the pluck release. As I discuss in Section 4.3.2, calculating string releases represents a surprisingly large proportion of the total calculations when simulating “real-world” music.

It is worth noting that Figure 2.19 contains unwanted peaks at ≈ 420 Hz and ≈ 840 Hz; using two (or even three) forces is not sufficient to completely eliminate the frequency we investigated with the “unit string”. However, those two peaks both have ≈ 75 dB less energy than the partial of slightly lower frequency, so they are not a serious concern.

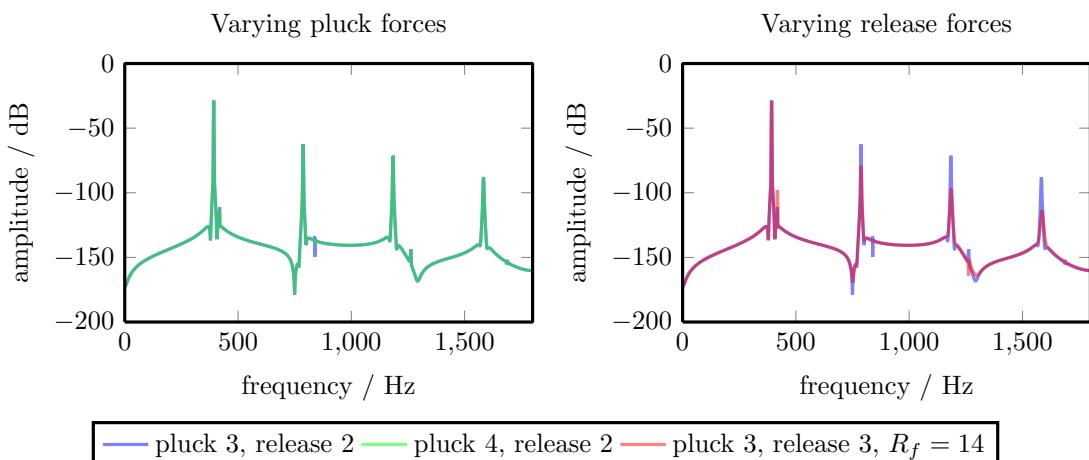


Figure 2.19: Comparison of varying numbers of forces for plucks. Violin G string, finger $x_f = 0.5L$, pluck $x_p = 0.2L$, $R_f = 30$ unless otherwise stated. Left: no significant difference between 3 and 4 forces during the plucking stage. Right: differences between 2 and 3 plucks during the release stage; higher damping is expected due to having three damped springs instead of two, so R_p was reduced in order to keep the magnitude of the first peak the same for the 2- and 3-force versions.

Audio 2.12: Pluck with three forces, release with two forces.

<http://percival-music.ca/dissertation/a.2.12.pluck-release-force-two-violin-g.wav>

Audio 2.13: Pluck with four forces, release with two forces.

<http://percival-music.ca/dissertation/a.2.13.pluck-release-force-four-violin-g.wav>

Audio 2.14: Pluck with three forces, release with three forces.

<http://percival-music.ca/dissertation/a.2.14.pluck-release-force-three-violin-g.wav>

2.3 Design decisions and consequences for bowing actions

The main method of playing violin is by bowing the string, so it is important that it work well. There are three issues to note with the bowing: Placing the bow at a “strongly rational” position, not allowing the string to move faster than the bow with constant bowing parameters, and allowing the bow slip-state to skip over the stick-state during changes of bow velocity. To investigate the effects of certain bowing parameters, I turn to two widely-used (Tzanetakis 2002, Peeters 2004) audio features: The spectral flatness measure and spectral centroid.

The Spectral Flatness Measure (SFM) is a measure of the “tonality” of a signal — that is, how much energy is concentrated in narrow peaks. White noise has a SFM of 1.0, while values close to 0.0 indicate a mixture of sine waves. Formally, given the Discrete Fourier Transform (DFT) bins $X[k]$ of a signal $x[n]$, the SFM is the geometric mean of the power spectrum divided by the arithmetic mean,

$$\text{SFM}(X) = \frac{\exp\left(\frac{1}{N} \sum_{n=0}^{N-1} \ln(|X[n]|^2)\right)}{\frac{1}{N} \sum_{n=0}^{N-1} |X[n]|^2} \quad (2.51)$$

The Spectral Centroid (SC) is the amplitude-weighted mean of frequencies, and is associated with the perceived “brightness” of audio. Given the DFT bins $X[k]$ and centre frequency of bins $X_f[k]$,

$$\text{SC}(X_a, X_f) = \frac{\sum_{n=0}^{N-1} |X[n]| \cdot X_f[n]}{\sum_{n=0}^{N-1} |X[n]|} \quad (2.52)$$

It is common to restrict $X[k]$ to a particular band of frequencies in both the SFM and SC. I will be restricting SFM to the frequency band of 20 Hz to $4.5\omega_0$, where ω_0 is the fundamental frequency of the string. I restrict the spectral centroid to the range $0.9\omega_0$ to $1.9\omega_0$ as shown in Figure 2.20. Finally, I define the normalized spectral centroid difference (SCN),

$$\text{SCN}(X_a, X_f, \omega) = \frac{|\omega - \text{SC}(X_a, X_f)|}{\omega} \quad (2.53)$$

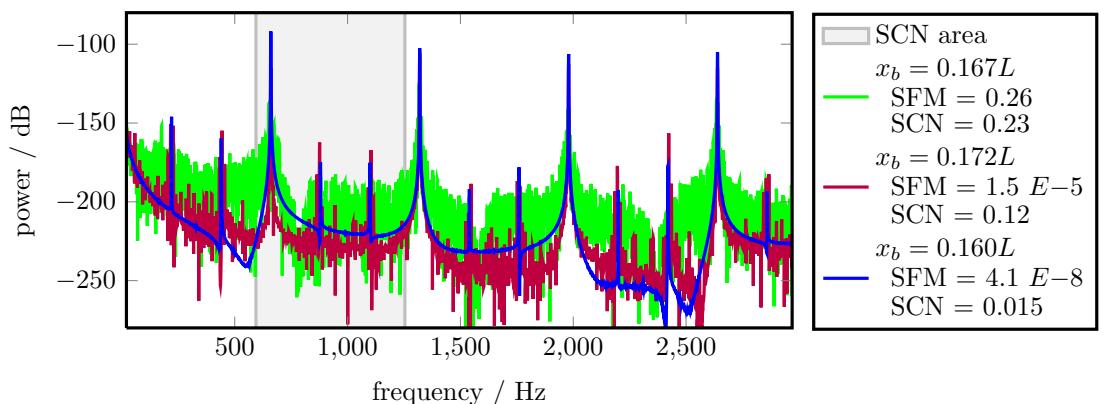


Figure 2.20: Frequency bands of bowing position x_b analysis, violin E string with $f_s = 88200$ Hz. Finger $x_f = 0$, force $F_b = 0.482$ N, velocity $v_b = 0.1$ m/s. The string was simulated for 1 second to allow initial transients to settle, then simulated for a further 1 second to create the analysis data. Audio 2.15: Bowing at $x_b = 0.160L$, $x_b = 0.167L$, and $x_b = 0.172L$.

<http://percival-music.ca/dissertation/a.2.15.bow-s3-p0.160-f0.482-v0.100.wav>

<http://percival-music.ca/dissertation/a.2.15.bow-s3-p0.167-f0.482-v0.100.wav>

<http://percival-music.ca/dissertation/a.2.15.bow-s3-p0.172-f0.482-v0.100.wav>

2.3.1 “Strongly rational” positions in bowing

“Strongly rational” positions pose a problem for bowing, although not quite as much of a problem as they posed for plucking. Although the left-hand finger x_f can be at positions such as $\frac{1}{2}L$ or $\frac{2}{3}L$, the bow’s placement x_b does not include any “very strongly rational” position in normal playing. Expert violinists keep their bow between $0.015L$ and $0.18L$ (Schoonderwaldt 2009), while I estimate that violin students’ bow position ranges from $0.06L$ to $0.21L$ and cello students’ bow position ranges from $0.05L$ to $0.16L$. As such, the bow position will never be on a node of any of the first four modes. However, “moderately strong” rational positions such as $\frac{1}{6}L$, $\frac{1}{7}L$, and $\frac{1}{8}L$ fall within the expected range of bowing of beginners and experts alike.

Analyzing simulations in which the bowing parameters (x_b , F_b , and v_b) are varied shows a clear pattern of “rational” bowing positions affecting the output. Figure 2.21 shows a few examples; similar experiments on other strings reveals the same modal-position-dependent behaviour, albeit slightly less prominent on strings with a lower fundamental frequency. All plots show “ridges” along bow positions x_b of $\frac{1}{5}L$, $\frac{1}{6}L$, $\frac{1}{7}L$, etc. Surprisingly, there are also ridges at $x_b = 0.182L$ and $x_b = 0.154L$; these correspond to $\frac{2}{11}L$ and $\frac{2}{13}L$ respectively.

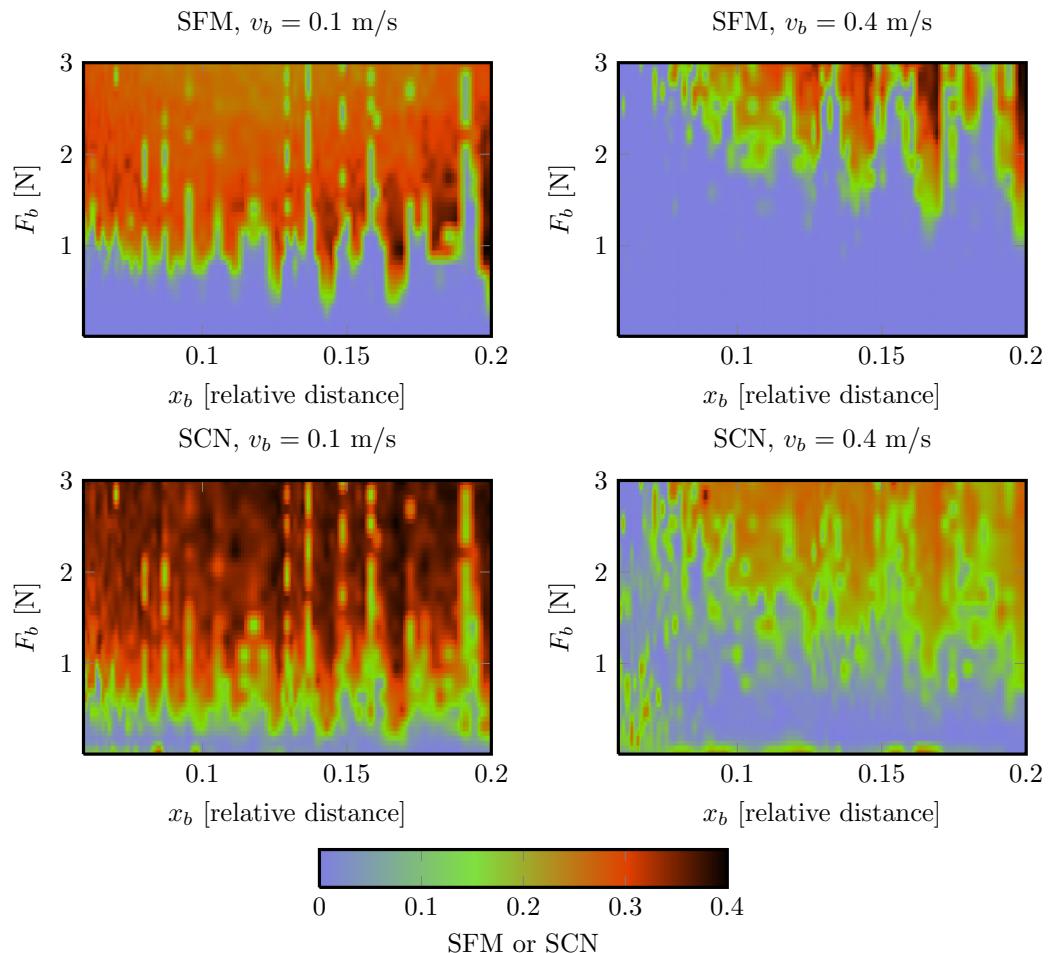


Figure 2.21: Effect of rational bow positions x_b on violin behaviour, violin E string at $f_s = 88200$. $v_b = 0.1$ or 0.4 m/s, while F_b and x_b are varied more gradually. The string was simulated for 1 second to allow initial transients to settle, then simulated for a further 1 second to create the analysis data.

It should be clarified that there is nothing wrong with the sound varying depending on bow position; the effect of bow position on sound was examined by all studies of violin mechanics from (Helmholtz 1895) onwards (see Section 1.2.1 for more information). However, real-life violins show a gradual change of audio based on bowing position, without sharp ridges at strongly rational positions. Real-life bows are approximately 1cm wide, and thus even if one edge of the bow is exactly at a node, the rest of the bow width will still excite that mode.

The effects of modal positions could be reduced by simulating the bow width with two or more bow positions, as I did for plucking the string. However, this would drastically increase the complexity of the bowing algorithm. Plucking a string involves a set of linear equations; adding an extra equation or two merely increases the size of the matrix to solve. However, bow friction is a non-linear equation; if we added an extra bow position, the direct solution of the two-equation bow system described in Section 2.1.3 would not be sufficient to solve the new three-equation system. Rather, I would need to solve the equations using a non-linear numerical algorithm such as the iterative Newton-Raphson method. The system of equations becomes even more complicated when one considered the changing bow-state. Adjusting the algorithm to solve such a system is possible and should be done for any scientific examination of the string, but it would significantly increase the computations required for each time step and may result in the simulation being unable to simulate a string in realtime on current consumer computers.

For this reason, I accept this unfortunate rational-position-dependent behaviour of the model. There are some mitigating factors, however. First, violinists are accustomed to adjusting their bowing to avoid odd problems; the space of bowing parameters which lead to “good sound” is quite irregular (Serafin 2004, Guettler 2002, Schoonderwaldt 2009). I admit that this model is “more irregular” than a real violin, but it is a question of degree, not kind. It is quite plausible that a violinist performing music on a haptic bowing interface with audio from this simulation would not notice the particular positions which produce sub-optimal audio; the violinist could simply adjust the bowing parameters subconsciously until a good sound was reached⁵. Second, to anticipate later material on the virtual musician in Section 5.1.3, my virtual musician will have the bow position x_b set manually by the teacher. By selecting bowing positions which do not correspond to strongly rational points, I can avoid this problem.

2.3.2 String moving faster than the bow

The model makes two assumptions which are decent “first approximations”, but which can cause problems in certain situations. These assumptions are that that bow friction can be expressed with the hyperbolic friction curve with hysteresis, and that external forces are constant throughout the time interval dt . One problem is that if the sampling rate is not high enough, the string can move faster than the bow with constant bowing parameters. This is not a problem in general; if we change bowing parameters (e.g., suddenly stopping the bow) then we should expect that the string will not be instantly limited by the bow speed. But when a string is bowed from rest with constant bowing parameters, we do not expect the string to move faster than the bow.

⁵Note that the problems (i.e. very high values of SFM and SCN) occurred when the bowing parameters were constant; real musicians accelerate the bow from rest, which has a much wider range of bowing parameters leading to Helmholtz motion (Guettler 2002). Any number of small changes to bowing parameters can lead to the string reaching steady-state Helmholtz motion; Figure 2.21 shows that the strongly rational bowing positions x_b have different behaviour from other positions, not that it is impossible to *maintain* Helmholtz motion in a strongly rational position if it was reached with a different set of bowing parameters.

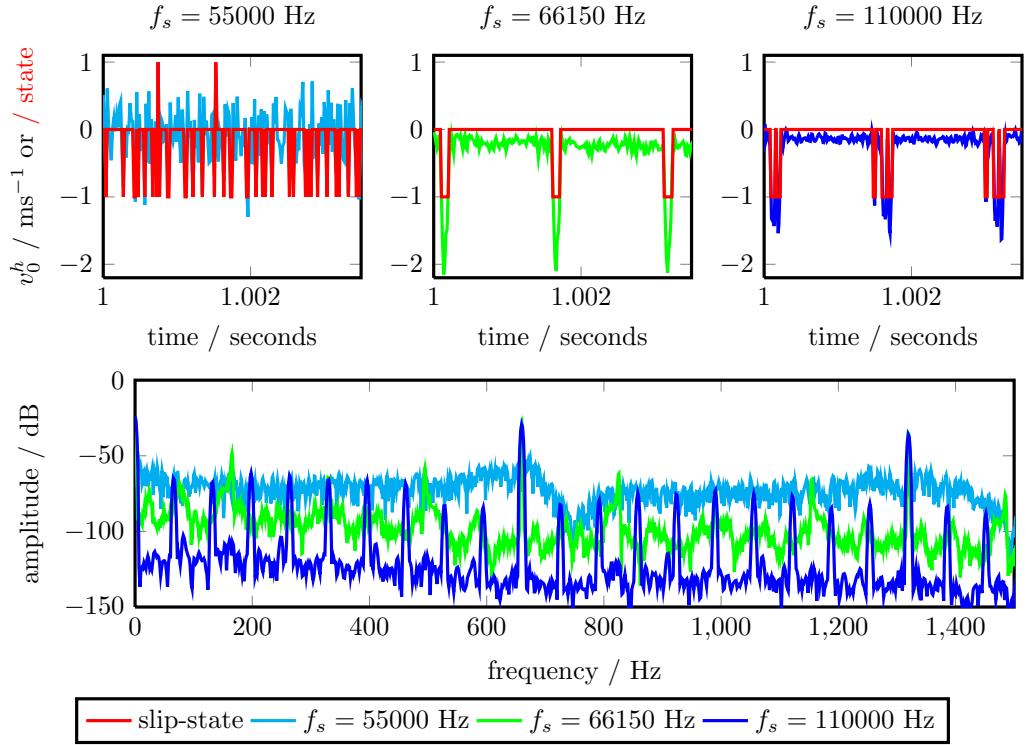


Figure 2.22: Low sampling rate causing the string to move faster than the bow. Violin E string, no finger ($x_f = 0$), bowing parameters $x_p = 0.12L$, $F_b = 0.25$ N, $v_b = 0.1$ m/s. The string was bowed for 1 second to allow it to settle, then 1 additional second to gather spectral data. The red line indicates the bowing slip-state, ranging from negative slipping (-1), sticking (0), or positive slipping (+1). The middle plot shows Helmholtz motion (one slip per cycle of 660 Hz or 1.5 ms); the left plot shows chaotic behaviour; the right plot shows undesirable “multiple slipping” behaviour which results in strong lower harmonics shown in the spectrum.

Audio 2.16: Positive bow slipping at $f_s = 55000$ Hz.
<http://percival-music.ca/dissertation/a.2.16.bow-slip-single-freq-low-violin-e.wav>

Audio 2.17: No positive bow slipping at $f_s = 66150$ Hz.
<http://percival-music.ca/dissertation/a.2.17.bow-slip-single-freq-moderate-violin-e.wav>
 Audio 2.18: No positive bow slipping at $f_s = 110000$ Hz.
<http://percival-music.ca/dissertation/a.2.18.bow-slip-single-freq-high-violin-e.wav>

This problem is shown in Figure 2.22 with the positive slips in $f_s = 55000$ Hz. How do these arise? The highest modal frequency is 26409.6 Hz, so the Nyquist limit is not a concern. The equations used for the bowing simulation contain three variables which represent the bow’s internal state: v_0^h , y_1^h , and v_1^h . In this case, there is no left-hand finger on the string, so y_1^h and v_1^h are both 0. With constant bowing parameters, the bow’s slip-state depends entirely on v_0^h and the current slip-state (the hysteresis). A transition to a positive slip-state will occur when the intersection of F_{modal} (2.34) and F_{friction} (2.36) falls at the very bottom of the vertical “sticking” line:

$$F_{\text{modal}} = D_1(v_b + \Delta v - v_0^h) + D_2 y_1^h + D_3 v_1^h = -F_b \mu_s = F_{\text{friction}} \quad (2.54)$$

Considering only the case of an open string ($x_f = 0$, therefore $y_1^h = v_1^h = 0$), this simplifies to

$$v_0^h = F_b \mu_s B_{00} + v_b \quad (2.55)$$

An example of friction behaviour and slip-states is shown in Figure 2.23 and Table 2.6. Recall that we must find the intersection of a diagonal F_{modal} line with the F_{friction} curve. The slope of F_{modal} is D_1 , which in the case of having no left-hand fingers on the string is simply $\frac{1}{B_{00}}$. For a given sampling rate f_s , the string's v_0^h will cause the diagonal F_{modal} line to shift along the x axis, but the slope of the line will remain constant.

Imagine sliding a pen horizontally along Figure 2.23, starting from the left-hand side of the graph and keeping the pen parallel to the green diagonal lines corresponding to the $f_s = 55000$ Hz. As long as the pen remains to the left of the green area, the string will be in a negative slipping state; as soon as the pen falls within the green area, it will stick. Now put the pen (still parallel to the green lines) back on the left-hand side of the graph, but instead of sliding it gradually, move it along the x axis in discrete non-uniform steps. Provided that the discrete steps are small enough, the pen will land within the “sticking area” and the string equations will behave as expected. However, suppose that the discrete steps were quite large. The pen could jump over the sticking region entirely, and land in the positive slipping area. That is the problem which was shown Figure 2.22.

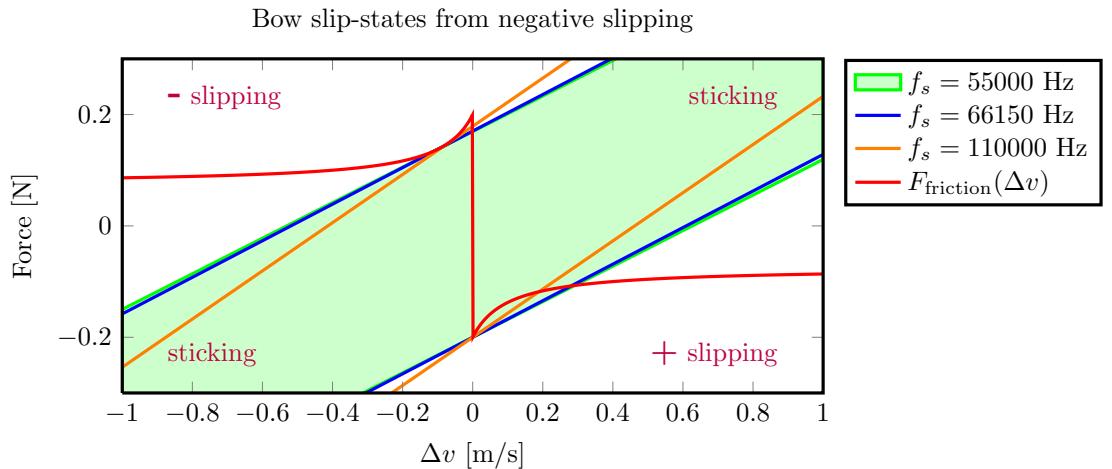


Figure 2.23: Friction slip-states transitions from a negative slip-state. Violin E string, no finger ($x_f = 0$), bowing parameters $x_p = 0.12L$, $F_b = 0.25$ N, $v_b = 0.1$ m/s. The full “sticking” region is shown for $f_s = 55000$ Hz; for higher frequencies only the lower and upper boundaries are shown. Note that the blue lines for $f_s = 66150$ Hz have a slightly steeper slope than the green lines for $f_s = 55000$ Hz. Numerical data is given in Table 2.6.

f_s	B_{00}	$\min v_0^h$	$\max v_0^h$	$\text{width } v_0^h$
55,000	3.13	-0.43	0.73	1.16
66,150	3.05	-0.42	0.71	1.13
111,000	2.31	-0.31	0.56	0.88

Table 2.6: Limits on v_0^h for transition to a sticking slip-state when the string is in a negative slip-state. Violin E string, no finger ($x_f = 0$), bowing parameters $x_p = 0.12L$, $F_b = 0.25$ N, $v_b = 0.1$ m/s. The min/max/width refers to the bounds of the sticking region. Graphical data is shown in Table 2.6.

As previously mentioned, the maximum v_0^h which leads to a stick-state is given in (2.55). The minimum v_0^h occurs when F_{modal} is tangent to F_{friction} . If the string is currently in the negative slipping state, the slopes are

$$\frac{dF_{\text{friction}}}{d\Delta v} = \frac{F_b\mu_e(\mu_s - \mu_d)}{(\mu_e - \Delta v)^2} \quad (2.56)$$

$$\frac{dF_{\text{modal}}}{d\Delta v} = D_1 \quad (2.57)$$

We let (2.56) equal (2.57) and solve for Δv , then take the minimum. Recall that $D_1 > 0$ (see Appendix A).

$$\begin{aligned} \Delta v &= \min \left(\mu_e + \frac{\sqrt{F_b\mu_e D_1(\mu_s - \mu_d)}}{D_1}, \mu_e - \frac{\sqrt{F_b\mu_e D_1(\mu_s - \mu_d)}}{D_1} \right) \\ &= \mu_e - \left| \frac{\sqrt{F_b\mu_e D_1(\mu_s - \mu_d)}}{D_1} \right| \\ &= \mu_e - \frac{\sqrt{F_b\mu_e D_1(\mu_s - \mu_d)}}{D_1} \end{aligned} \quad (2.58)$$

By assumption the string is in a negative slipping state, and any sensible⁶ set of bowing parameters will lead to $\Delta v < 0$, which can be substituted into the result after solving $F_{\text{modal}} = F_{\text{friction}}$ for v_0^h ,

$$v_0^h = \frac{((\mu_e - \Delta v)v_1^h D_3 + y_1^h D_2(\mu_e - \Delta v) + D_1((\mu_e - \Delta v)(v_b + \Delta v)) + F_b(\mu_d\Delta v - \mu_s\mu_e)}{D_1(\mu_e - \Delta v)} \quad (2.59)$$

Applying the simplification of no left-hand finger,

$$v_0^h = \frac{B_{00}F_b(\mu_s\mu_e - \mu_d\Delta v) - (\mu_e - \Delta v)(v_b + \Delta v)}{\mu_e - \Delta v} \quad (2.60)$$

Increasing the sampling rate decreases the width of the sticking region, but also decreases the amount that v_0^h changes between time steps. It should be noted that there is no linear relationship between either of these factors. Given constant bowing parameters, the width of the sticking region depends solely on B_{00} as seen in (2.55) and (2.60), which in turn depends entirely on Y_{3n} from Table 2.2. The amount of change between $v_0^h(t_0)$ and $v_0^h(t_1)$ depends on $\dot{a}_n(t_0)$, $\dot{a}_n(t_1)$, and Y_{3n} . In turn, $\dot{a}_n(t_1)$ depends on almost all parts of the physical model, so an analytic prediction about the effects of increasing f_s would be very challenging. Empirically, increasing the sample rate reduces the number of positive slips; for example, simulating the violin E string at 88200 Hz produces no positive slips. Avoiding positive slips (when bowed in a positive bow direction) will be the main consideration behind choosing a sampling rate for each string in Section 3.5.2.

⁶With very odd bowing parameters (i.e. $\mu_e \geq \frac{\sqrt{F_b\mu_e D_1(\mu_s - \mu_d)}}{D_1}$), there may not be an acceptable solution, in which case the minimum v_0^h is calculated based on the upper “sticking” line and will be $v_0^h = -F_b\mu_s B_{00} + v_b$ for a string without a left-hand finger.

2.3.3 “Safety valve” changes of slip-state

The sampling rate of each string is chosen to avoid undesirable slip-states when starting from rest, but there are two remaining concerns. First, that it is necessary to allow positive slip-states; and second, that it is necessary to allow the string to skip over the sticking state. The necessity of both elements of the model arise from circumstances associated with changing bowing parameters.

Consider a string which establishes Helmholtz motion with a moderately large bow velocity v_b and force F_b , then suddenly the bow is lifted away from the string ($F_b = 0$). The string will vibrate freely; v_0^h will fluctuate between positive and negative values and will gradually decay according to the modal decay rates. Now suppose that rather than removing the bow, we instead drastically decrease v_b and F_b but keep them non-zero. Recall that the lower and upper bounds of the vertical “sticking” portion of F_{friction} in (2.36) are $-F_b\mu_s$ and $F_b\mu_s$; reducing F_b will reduce the region of v_0^h which leads to a sticking state. Such an example is shown in Figure 2.24.

What should be done if v_0^h “skips over” the sticking region and lands in the positive-slipping region? Although I have chosen sampling rates for each string which avoid this problem for constant bowing parameters, when we allow changing bowing parameters then v_b and F_b can be arbitrarily small, leading to an arbitrarily narrow sticking region. As long as external forces are assumed to be constant for the time interval dt , it is not possible to avoid the problem of v_0^h being too large for the normal sticking area. One solution is to forbid any positive slips: The string is only allowed to stick or slip in the opposite direction of the current bow velocity v_b . This method was used in (Demoucron 2008), but examining Figure 2.24 suggests that this can lead to undesirable behaviour. If the string is vibrating normally according to the friction states in the left-hand plot, then it is likely that the F_{modal} diagonal line will be oscillating with a x -intercept above 0. When the bowing parameters change, the string’s vibrations will not instantly cease. When the bowing parameters change and we switch to the friction states in the right-hand plot, any F_{modal} line with a x -intercept above 0 will result in a very large negative force being applied to the string. This can lead to “choking” the sound when v_b and F_b are suddenly reduced, as shown in Figure 2.25.

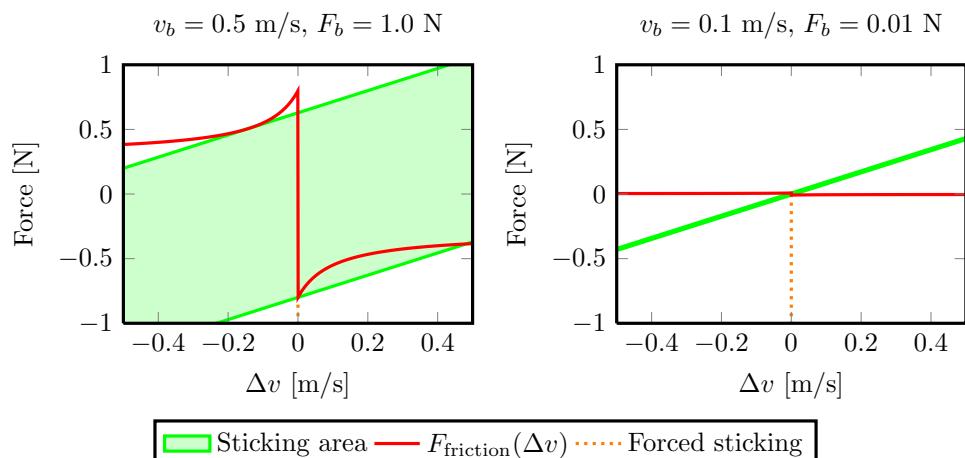


Figure 2.24: Friction slip-states for a bowing change. Violin G string at $f_s = 44100$ Hz, no finger ($x_f = 0$), bow position $x_b = 0.12L$. The dotted orange line indicates the force which the bow exerts on the string if positive slips are forbidden — instead of finding the intersection of F_{modal} with F_{friction} , imagine that the F_{friction} curve does not exist for $\Delta v > 0$ and instead follows the dotted orange line.

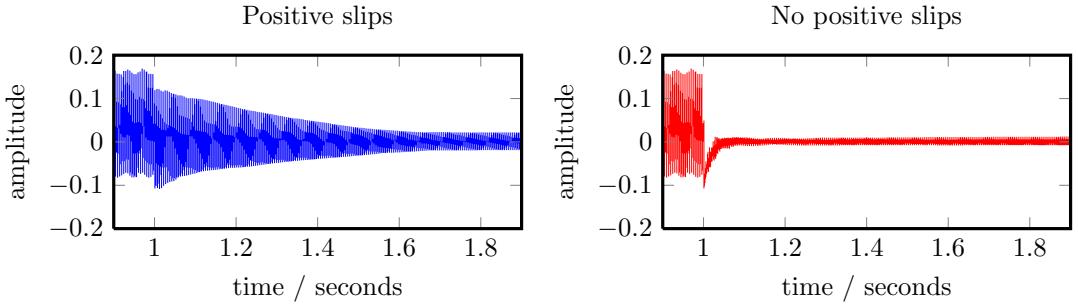


Figure 2.25: Effect of forbidding positive slips. Violin G string at $f_s = 44100$ Hz, no finger ($x_f = 0$, bow position $x_b = 0.12L$). String was bowed for 1 second with $v_b = 0.5$ m/s, $F_b = 1.0$ N, then bowed for 2 more seconds with $v_b = 0.1$ m/s, $F_b = 0.01$ N. In order to ensure that both simulations began from the same point at 1 second, the random portion of the hyperbolic curve was disabled ($\mu_e = \mu_c$). When positive slips are allowed, the string's vibrations are gradually cut off. When positive slips are forbidden, the string's vibrations are abruptly and unnaturally cut off.

Audio 2.19: Change of bowing parameters with positive slips allowed.

<http://percival-music.ca/dissertation/a.2.19.bow-slip-change-both-violin-g.wav>

Audio 2.20: Change of bowing parameters with positive slips forbidden.

<http://percival-music.ca/dissertation/a.2.20.bow-slip-change-negative-only-violin-g.wav>

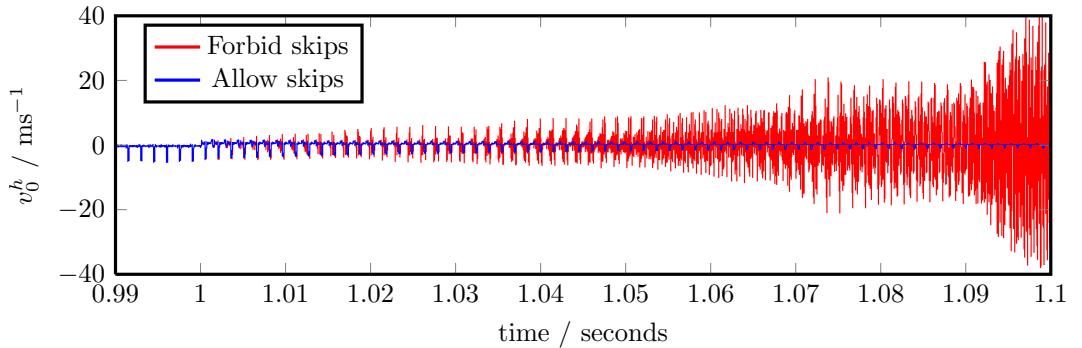


Figure 2.26: Effect of forbidding skipping over the stick-state. Violin E string at $f_s = 88200$ Hz, no finger ($x_f = 0$, bow position $x_b = 0.12L$). String was bowed for 1 second with $v_b = 0.5$ m/s, $F_b = 1.0$ N, then bowed for 0.15 (for no-skips) or 2.0 more seconds with $v_b = 0.1$ m/s, $F_b = 0.01$ N. The random portion of the hyperbolic curve was disabled ($\mu_e = \mu_c$).

Audio 2.21: Change of bowing parameters with skipping over stick-state.

<http://percival-music.ca/dissertation/a.2.21.bow-slip-change-both-safety-violin-e.wav>

Audio 2.22: Change of bowing parameters forbidding skipping over stick-state. Caution: very loud audio at 1 second.

<http://percival-music.ca/dissertation/a.2.22.bow-slip-change-no-skips-safety-violin-e.wav>

For this reason, I allow positive slipping states. The second question is whether I should allow the string to “skip” directly from negative slipping to positive slipping, or whether I should force the string to have at least 1 time-interval of sticking between the two slip-states. Recall that the model assumes that the external forces have a constant value through the time interval dt , and that F_{friction} changes drastically depending on the slip-state. For most strings, there is no great difference between allowing or forbidding skips, but for thin and high-frequency string such as the violin E string, forbidding the string to skip directly from negative to positive slipping leads to unstable (exponentially increasing) behaviour as shown in Figure 2.26.

2.4 Final remarks on the physics

This chapter has described the equations comprising my physical model and examined the consequences of using those equations. As discussed in Section 1.3.2, this model is far from the most realistic description of violin mechanics known to scientists; however, it provides audio of sufficiently high quality that it will suffice for my purpose. I will review the mechanics which are not covered by my model, and then summarize the actions that the model can simulate.

The major research contributions of this chapter are:

- Added plucking actions to Demoucron’s model, and showed why at least two point forces for each finger are necessary when using a modal simulation. This is discussed in Sections 2.1.2 and 2.2.
- Fixed Demoucron’s bowing model to avoid an unstable system when simulating the violin E string with experimentally measured modal damping factors. Two problems with the bowing model were demonstrated but not fully resolved: Odd behaviour can occur at insufficiently high sampling rates, and at bow positions which are “strongly rational”. I mitigate these problems by choosing the sampling rate multiplier M_f for each string (Section 3.5.2) to reduce such oddities, and avoid “strongly rational” bow positions for the virtual violinist (Section 5.2). This is discussed in Sections 2.1.3 and 2.3.
- Added haptic output to the model as discussed in Section 2.1.4. This allows the model to be used in a force-feedback system for human interaction, but my main goal is to provide additional information for the machine learning of the virtual violinist (Section 5.2).

2.4.1 Model simplifications

I adopted Demoucron’s model based on my subjective judgement of the quality of audio I considered necessary for my virtual violinist. Other researchers may wish to work with a more accurate model, either by modifying this model or by choosing a different method of synthesis entirely. The choice of model depends on a combination of the intended use of the simulation, the amount of processing time available, and the amount of researcher time available to spend on physics or programming. An excellent review of bowed-string mechanics is given in (Woodhouse & Galluzzo 2004), while (Demoucron 2008) contains a list of criticisms of his model which apply to this model as well.

In the model described in this chapter, the string is simulated as a set of 1-dimensional vibrating transverse modes, each of which has a constant decay rate. However, real-world strings vibrate in many more directions: Two transverse dimensions (parallel and normal to the bow), torsional motion, and longitudinal motion. Most strings (other than the violin E string) are not uniform materials, as they consist of a thin layer of metal wound around a nylon-like core; this introduces additional internal friction.

Placing the string on an instrument body adds another set of known mechanics which are not included in this model. The instrument body itself is a vibrating non-linear system. In a real instrument, energy is transferred between the instrument body and the strings at the bridge and nut; in the model, energy is only transferred from the string to the bridge in one direction. In addition, the bridge (as distinct from the instrument body) is another vibrating system.

In the model, the left-hand finger is assumed to be one point (during bowing) or two points (when the bow is released) acting as damped springs. In real life, a finger has a wide force profile, exerting force on the string at all points between the two edges. In addition, the left-hand finger on the violin or viola is placed on the fingerboard in a different orientation than a left-hand finger on the cello when the cellist is playing in the normal (low) playing range — but when the cellist plays in “thumb position” (a relatively advanced technique), the cellist’s fingers act like violinist fingers, while the cellist’s thumb has a completely different profile again. Finally, musicians often have up to four fingers pressed against the string with varying forces, not a single finger as in the model.

The bow-string interaction in the model uses the old hyperbolic friction curve with hysteresis. More accurate models of this interaction exist, such as a double-exponential model or a temperature-sensitive model which accounts for the frictional force generating heat which partially melts the rosin on the bow, thereby changing the friction characteristics. In addition, the model assumes that the bow has constant friction characteristics, whereas a real bow’s friction varies based on the amount of rosin which is at each point along the string: Musicians do not apply rosin to the bow in a perfectly uniform manner, and even if they did, after playing the instrument for a few hours the amount of rosin left on the bow would vary based on the parts of the bow which were used while playing the instrument. Finally, the bow is assumed to be an infinitely narrow point which only excites the string in one direction; a real bow has width, and would excite multiple directions of vibration. Unfortunately, modelling the bow as two points is much more complicated than adding an extra finger point force. Assuming the two bow point forces are independent, in the worst case a quartic equation ($x^4 + Ax^3 + Bx^2 + Cx + D = 0$) must be solved for each time step. This would require an iterative solution, greatly reducing the speed of simulation. If the bow width is not modelled as independent forces, then an entirely different bowing model will be needed. The bow’s effect on the string is in turn affected by the bow’s tilt (angle around the length axis) and skew (angle between bow and the bridge). The control of bow tilt is explicitly taught to students with a few years of experience, while control of skew is taught to students with ten years of experience.

The bow itself is not modelled at all; a real bow is a vibrating system. Expert violinists consider the bow to be extremely important for the production of good tone, and will pay up to half of the price of an instrument body for the bow. Physically speaking, each bow will have a different centre of gravity and mass, but the main difference which musicians seek is the behaviour of vibrations in the bow, and how those vibrations aid (or foil) their attempts to achieve Helmholtz motion in the string. In addition to the bow’s vibrations, the bow hair deforms and stretches when it is pressed against the string. In terms of the model described in this chapter, the deformations in bow hair and vibrations in the bow would cause F_b to vary enormously even if the user was exerting the same amount of pressure with her right hand. F_b has a huge effect on the friction slip-states, and would therefore greatly alter the range of bowing parameters which can lead to Helmholtz motion.

Finally, it would be interesting to model other elements which affect human performance. The violin and viola are held on the musician’s left shoulder, requiring a different bow angle in order to play each string; on most strings gravity will pull the bow directly towards the string (increasing F_b), although the E string is at a slight angle. In contrast, the cello is held at an angle in a sitting position; gravity pulls the bow closer to the bridge (decreasing x_0 , with perhaps a slight increase of F_b). The musician must use alter her right hand behaviour to counter gravity depend on which instrument she is playing. Human biomechanics plays a large role in how musicians perform.

However, despite all these simplifications, the model produces sound which I consider acceptable for my virtual musician. The physical model described by (Demoucron 2008) provided a strong starting point, and after the alterations described in Section 2.2 and Section 2.3 it serves my purpose. Future researchers interested in computer musical expression and control may opt to use the model as described or extend it in certain areas; researchers interested in a scientific examination of musical instruments are advised to use a different model. If widespread use of the model is desired (e.g., producing commercial music CDs, submitting synthesized performances of music to a composition competition, using the model in high school physics classes), then more stringent listening tests should be performed. A double-blind listening test should be performed with the target audience (be they musicians, researchers, or the general public) to compare the perceived quality of the synthesized audio and real string instruments.

2.4.2 Summary of available actions

The model provides these actions:

Left-hand finger: place a finger on the string, specifying the distance from the bridge x_f and the spring strength K_f . The spring strength will generally be the default value (10^5), but it may be reduced in order to play a harmonic (required value depends on x_f and the bowing parameters).

Bow: place a bow on the string with bow-bridge distance x_b , force F_b , and velocity v_b . Optionally, the bow acceleration a_b and target velocity v_b^t may be set.

Pluck: pluck a string at distance from the bridge x_p by with the desired pulling distance y_p^d .

Wait: simulate m samples.

All musical performance with *Vivi, the Virtual Violinist* will be reduced to a combination of these four actions, along with the model's output of an audio $A[t]$ and haptic $H[t]$ signal.

If future researchers desire to use *Vivi* with a different synthesis engine (be it physical modelling or sampling synthesis), this list defines a relatively small set of instructions which the engine would be required to support.

Chapter 3

Constants for Physical Modelling

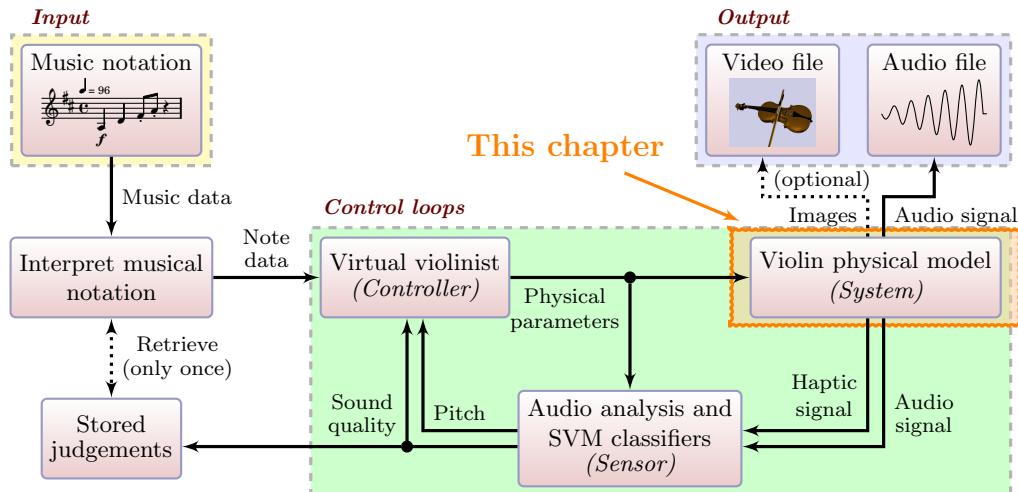


Figure 3.1: Physical modelling in context.

When scientists create physical models to investigate acoustic properties of musical instruments, they generally measure and experiment with a single instrument. This is quite appropriate from a scientific viewpoint; if the model behaves well with a string of length 0.39 m, then it will probably behave similarly well with a string of length 0.52 m. However, from a musical standpoint it seems a waste to imitate one single instrument. One of the great advantages of physical modelling compared to sample-based synthesis (including spectral modelling synthesis) is that perceptually distinct instruments can be created merely by changing a few constants. It would greatly increase the musical usefulness of the program if it could simulate a string quartet or even octet.

This is particularly relevant to my research: the intention is that *Vivi, the Virtual Violinist* is able to learn to play any bowed string instrument. Such a claim could be supported by demonstrating the ability to perform on two different instruments (e.g., violin and cello), but the generality of *Vivi* would be much better supported if I demonstrated the ability to perform on many instruments. With those two goals in mind (musical use, and demonstrating the flexibility of my intelligent feedback control), I have gathered physical constants allowing me to simulate multiple distinct instruments. In particular, I have measured 10 instruments which were easily available to me: 5 violins, 2 violas, and 3 cellos.

Although many physical constants required for this model can be found in the scientific literature

on acoustics (Rossing 2010, Jansson 2002, Pickering 1985), there are two areas which require physical experiments: Modal decay values r_n used in (2.5), and the instrument body responses. In addition, the string length and diameter can be measured with non-intrusive, non-destructive methods. In contrast, a string's linear density and Young's elastic modulus cannot be measured without damaging or destroying the string. Instrument strings have additional windings at one end of the string, and a ball at the other end. Measuring the linear density of the vibrating portion of the string would require cutting those ends of the string, which would render the string unusable.

Acoustics measurements on modal decay values are often investigated for guitar or piano (Lee et al. 2010, Woodhouse 2004), or discussed in general terms (Karjalainen et al. 2002). However, there is no published collection of string modal decay values for violin, viola, and cello. One possible reason for this could be the popularity of digital waveguide synthesis for bowed string instruments, which relies on string end reflection filters rather than modal decay values.

Although there has been some excellent work on simulating the instrument body as a vibrating system (Serafin 2004, Inácio et al. 2008), for simplicity I followed Demoucron's physical modelling algorithm which assumes that the body is a linear time-invariant system. The impulse response is much easier to measure than the constants used in mathematical instrument body simulations, yet the results are still quite credible. Although there is a great deal of literature on violin body impulse responses (Fritz et al. 2007, Rossing 2010, Türckheim et al. 2010), there is again a lack of accessible data (rather than mere plots) of those impulse responses.

This chapter can be divided into four sections:

1. Physical experiments: Section 3.1 describes the instruments and measures their lengths and diameters. Section 3.2 discusses experiments and analysis of modal damping values. Section 3.3.2 gives measurements of instrument body impulse responses.
2. Estimating remaining constants: I combine known ranges of physical constants from acoustics literature with my measured constants in Section 3.4.
3. Simulations: I perform simulations with the physical model to investigate the effects and trade-offs for a few remaining constants in Section 3.5. In particular, these values either have no real-world equivalent (e.g., the finite number of modes N , the sampling rate f_s , threshold for turning off the simulation A_{\min}) or it would be meaningless to use real constants (e.g., the finger constants damped spring constants K_f and R_f for a two-point finger).
4. Final remarks: I summarize this chapter, discuss possible improvements to the measurements, and present the actual constants used for the physical modelling.

3.1 Overview of instruments and strings

A brief description of all ten instruments measured is given in Table 3.1. No information about strings was accessible; all instruments had been owned by their present owners for at least one year and strings were not changed regularly. This situation differs from that commonly quoted in the literature, in which new strings were used.

The string lengths and diameters were measured and these values are given in Table 3.2. String lengths were measured with a measuring tape and are believed to be accurate to 1%. An error

in planning the experiments resulted in not having a calliper while in contact with eight of the instruments. String diameters were therefore estimated by taking digital photographs of each string next to a ruler, counting pixels between 2cm or 5cm in the ruler to find the scale, then counting pixels in the string diameter. Later, when I had access to a Vernier calliper, the “counting pixels” measurements were checked by comparing the measured diameters of the remaining two instruments (violin-II and cello-I) with the calliper and the photographs. The string diameters are believed to be accurate to 10%.

name	estimated value (CDN \$)	country	year	notes
violin-I	5,000	Canada	1994	
violin-II	2,000	Germany	1908	
violin-III	8,000	USA	1875	
violin-IV	500	China	(1970)	(b)
violin-V	1,000	Romania	2005	
viola-I	2,000	China	(1990)	(b)
viola-II	500	Germany	(1970)	(a) (b)
cello-I	6,000	Canada	1997	
cello-II	2,000	Czechoslovakia	(1900)	(b)
cello-III	1,500	Germany	1992	

Table 3.1: Overview of instruments tested. The estimated values are a personal guess, not an official appraisal; these are probably accurate to 50%. Values reflect the price of the instrument, without the bow, case, or any extras.

(a) the D string was broken, so the A string was used (tuned down a fifth). This is the usual practice of string musicians in need of a quick fix for a broken string with no replacement.

(b) very cheap instruments are not stamped with a maker’s mark and year. I have included an estimated date based on the condition of the instrument.

string	L (mm)	d (mm)	notes	string	L (mm)	d (mm)	notes
violin-E-I	322	0.30		viola-A-I	363	0.33	
violin-E-II	319	0.28		viola-A-II	369	0.32	
violin-E-III	321	0.26		viola-D-I	363	0.48	
violin-E-IV	321	0.29		viola-D-II	369	0.32	(a)
violin-E-V	324	0.27		viola-G-I	364	0.61	
violin-A-I	322	0.67		viola-G-II	369	0.62	
violin-A-II	320	0.58		viola-C-I	365	0.65	
violin-A-III	323	0.68		viola-C-II	371	0.82	
violin-A-IV	320	0.57		cello-A-I	665	0.68	
violin-A-V	324	0.59		cello-A-II	685	0.70	
violin-D-I	322	0.80		cello-A-III	687	0.71	
violin-D-II	321	0.71		cello-D-I	666	0.80	
violin-D-III	325	0.67		cello-D-II	684	0.93	
violin-D-IV	320	0.69		cello-D-III	687	0.83	
violin-D-V	324	0.71		cello-G-I	669	1.10	
violin-G-I	322	0.93		cello-G-II	684	1.00	
violin-G-II	321	0.71		cello-G-III	687	0.98	
violin-G-III	326	0.69		cello-C-I	669	1.40	
violin-G-IV	321	0.89		cello-C-II	685	1.40	
violin-G-V	323	0.69		cello-C-III	687	1.50	

Table 3.2: Measured string lengths and diameters.

(a) the viola-II D string was broken, so the viola-II A string was used instead.

3.2 String modal decays

Before discussing the experiment, I will briefly describe what I am measuring. Figure 3.2 shows examples of the spectral energy after plucking a violin E string, viola D string, and cello C string. Those strings were chosen to include the highest and lowest-frequency strings, with the viola string serving as a middle ground. The energy of certain frequencies (the modes) decreases down to the noise floor. This decrease (i.e. the slope of the peak's energy against time) is the modal decay. In general, higher-frequency modes decay more quickly, but both examples contain deviations from this general rule. As may be expected from casual listening of violins and cellos, the cello string decays much slower than the violin string.

There are three main ways that the string's decay can be measured, listed in descending order of accuracy. The first method is to capture the string's motion with an optical pickup (used in some electric guitars), by shining a light source from one direction and placing a photodiode or phototransistor on the other direction. This provides a very accurate and low-noise method of measuring the string's motion, but it requires additional hardware to perform the experiment. The second method is to use magnetic induction, either with a magnet pickup (as used in most electric guitars) or by connecting the ends of the string to an analogue-digital converter (ADC). The magnetic field will slightly dampen the string (thereby altering the string's behaviour, unlike the optical sensor), but it requires less specialized hardware than the optical sensor. The third method of estimating the modal decays is to record the audio output of the instrument. An audio recording is subject to the non-linear effects of the instrument body, as well as environmental noise unless the recording is made in an anechoic chamber. I chose to use magnetic induction.

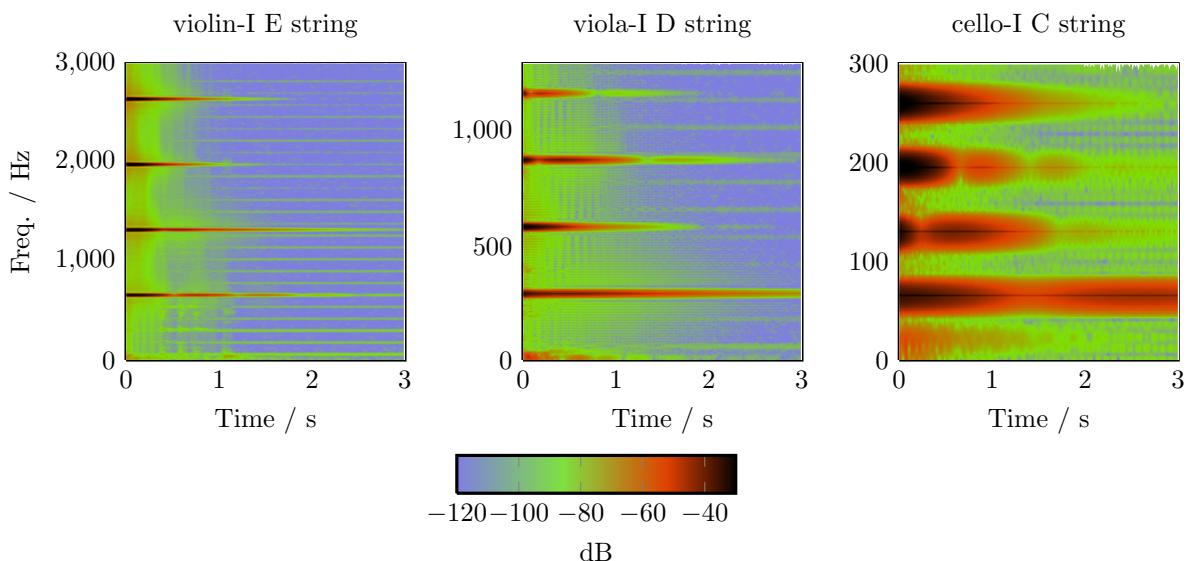


Figure 3.2: Spectrogram of plucking a violin E, viola D, and cello C string, first four modes. The violin E string modes decay much faster than the cello C string modes. All strings exhibit some beating, although this is much more pronounced in the C string.

Audio 3.1: Signal of current induced from violin E, viola D, and cello C strings.

```
http://percival-music.ca/dissertation/a.3.1.violin-e-i-01.wav
http://percival-music.ca/dissertation/a.3.1.viola-d-i-01.wav
http://percival-music.ca/dissertation/a.3.1.cello-c-i-01.wav
```

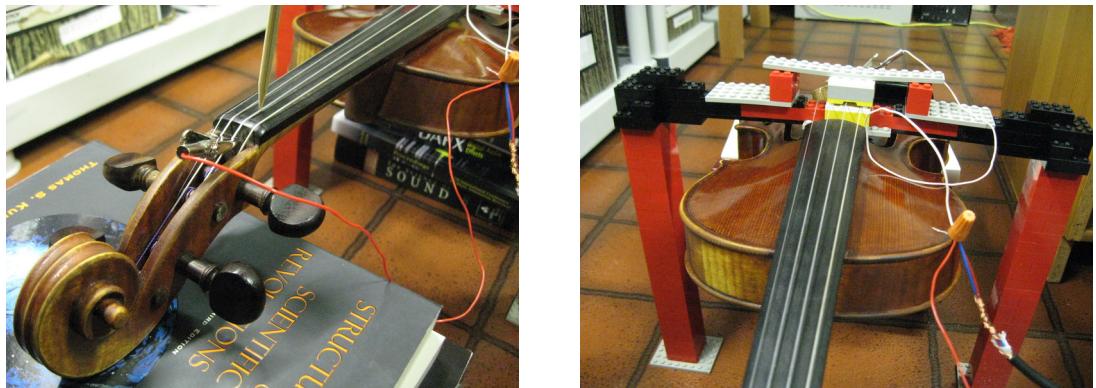


Figure 3.3: Photos of experimental setup for measuring decay of string velocity. Left: A wooden stick is used to “pluck” the string. Right: Plastic blocks hold the magnets in place with a frame.

3.2.1 Experimental procedure and analysis techniques

A strong and consistent magnetic field was created by placing two poles of 0.5 x 0.5x 0.125-inch neodymium magnets, residual flux density 12 kGauss, approximately 10mm on either side of a violin string. When the string moves through a uniform magnetic field, a voltage is induced. These voltages could be read with an oscilloscope, but given our goal of accessibility (Section 1.3), I instead use a semi-professional sound device¹. Such devices are optimized (modulo the cost of the unit) to have a good analog-digital converter (ADC) which produces 16-bit or 24-bit signals in the range of human hearing (20 Hz - 20,000 Hz, although typical values for music fall between 50 Hz and 10,000 Hz).

A sophisticated oscilloscope is unlikely to produce better (lower-noise) signals. Informal experimentation showed that the majority of noise arose from the string acting as an antenna and picking up electrical harmonics from AC power. Interestingly, acoustic noise was also a factor — a human speaking normally in the same room produced enough air pressure waves (amplified by the instrument body and carried through the instrument body and bridge into the string) to cause noticeable signals in the induced current. A more stringent scientific study of string decays should take additional measure to reduce radio frequencies and acoustic noise, but my intent here was merely to produce estimates which are sufficiently accurate to produce acceptable audio in the simulation.

In the experimental setup², the magnets were held in a frame constructed from Lego, which allowed the height of the frame to be adjusted to suit specific musical instruments. Wires from each end of the violin string were attached to the microphone input of a Tascam 122L sound device, which was then attached to a desktop computer with a USB cable. Before recording any plucks, 10 seconds of silence were recorded for each string in order to characterize the electrical noise from the Analog-Digital Converter (ADC), wires, alligator clips, and the string itself. The string was plucked 7 times with a wooden skewer approximately 1cm away from the nut, leaving at least 15 seconds for the pluck to decay between each pluck. After the 15 seconds had passed, the string was damped with a finger before beginning the next pluck. All data was recorded with 24-bit samples at 96 kHz.

¹With professional and semi-professional sound hardware, the ADC and DAC are almost always placed inside an external box connected via USB or firewire to reduce electrical noise. Occasionally the terms “sound interface” or “sound device” are used for these devices, but the term “sound card” is still often used.

²Dr. Paul Percival from Simon Fraser University (Canada) provided practical advice on reducing electrical noise and a few data analysis techniques.

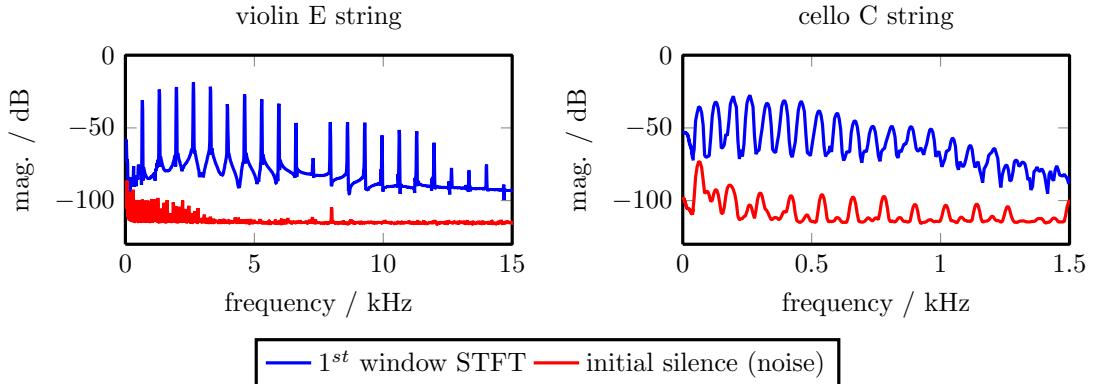


Figure 3.4: Pluck decay (first window) and noise of violin E and cello C strings. Note that these plots show different frequency ranges in order to focus on the most important peaks.

Spectral peak estimation and noise floor

The fundamental step in many digital signal processing algorithms is the Short-Term Fourier Transform (STFT). I used a hop size of 2048 and window size of 8192 (an overlap of 75%), with a Hamming window and a zero-padding factor of 4. The window size is large for the violin, but the cello C string has a fundamental frequency of ≈ 65 Hz. The sample rate was 96 kHz, giving the FFT a resolution³ of 11.7 Hz. Due to the width of the main window and side lobes resulting from the window function for the power STFT, using windows of size of 4096 (resolution 23.4 Hz) was not sufficient to adequately distinguish between modes on the C string. An example of the STFT for the first analysis window is shown in Figure 3.4.

To improve the maximum likelihood estimate of spectral peaks, I use the Quadratically-Interpolated FFTs (QIFFT) method (Abe & Smith 2004, Smith 2011). First, peaks in the log-magnitude array Y' are extracted. Then, given a peak at bin n , we calculate an estimated offset p from bin n , and an estimated magnitude y_p :

$$\alpha = Y'[n - 1] \quad \beta = Y'[n] \quad \gamma = Y'[n + 1] \quad (3.1)$$

$$p = \frac{\gamma - \alpha}{2(2\beta - \alpha - \gamma)} \quad y_p = \beta - 0.25(\alpha - \gamma)p \quad (3.2)$$

The initial 10 seconds of silence was processed in the same manner as the spectral peak estimation (i.e. split into windows of 8192 samples, Hamming windowed, STFT) to characterize the noise in the system. For each frequency bin, I define the noise level $N_{top}(\omega)$ as the 99th percentile of the magnitude of that bin throughout all 465 windows of the silence.

Two examples of these noise profiles are shown in Figure 3.4. The noise mainly follows the expected odd harmonics from 60 Hz AC power (in Canada), although a few odd “spikes” at 8 kHz (and its harmonics) and 24.7 to 24.9 kHz are present. I theorized that the wires in the setup (or even the string itself) acted as an antenna to pick up Very Low Frequency (VLF) signals⁴.

³Although zero-padding results in an FFT of size 32768 giving a resolution of 2.93 Hz, we must remember that zero-padding does not add any information to the signal; the “extra” resolution simply comes from interpolation. However, the “extra” resolution reduces the amplitude and frequency bias of the QIFFT method (Abe & Smith 2004).

⁴The origin of the 8 kHz signal is unknown; possibly due to the power supply breakthrough. The US Navy facility at Jim Creek (Oso, Arlington, USA; approximately 150 km away from Vancouver) is known to broadcast at 24.8 kHz for submarine communication with the Pacific fleet. <http://www.vlf.it/trond2/20-25khz.html> accessed 2012 Oct 08.

Segmenting multiple plucks

To evaluate the variation in modal decays, each string was plucked 7 times. To avoid excessive interaction with the computer, all plucks for each string were recorded in one audio file, then those recordings were automatically segmented into individual plucks. The segmentation program loads all samples in the recording. The median and standard deviation of the absolute value of all samples in each recording was calculated. The “trigger” of each pluck was defined as the median plus 7 times the standard deviation, provided that the onset occurs at least 6 seconds after the previous onset. The beginning of the pluck data was set to be the first zero-crossing after the trigger, while the duration of each pluck data is 10 seconds. After the automatic segmentation was performed, I listened to each segmented audio file to ensure that the segmentation was correct.

3.2.2 String inharmonicity and non-linearity

There are three factors which complicate the detection of string decays: String inharmonicity, beating, and non-linear mixing. String inharmonicity is clearly visible in Figure 3.5, comparing the predicted frequency⁵ f_n of mode n of an ideal flexible string with fundamental frequency f_0 ,

$$f_n = n f_0 \quad (3.3)$$

against the actual spectral data. This clearly is not an accurate prediction for the frequencies of peaks shown in Figure 3.5, so I turn to the string inharmonicity coefficient B (Fletcher et al. 1962, Jarvelainen et al. 2001). The frequency of partial f_n can be predicted by

$$f_n = n f_0 \sqrt{1 + B n^2} \quad (3.4)$$

In a circular beam of uniform material, the inharmonicity coefficient B can be predicted by

$$B = \frac{E \pi^3 d^4}{64 L^2 T} \quad (3.5)$$

However, we cannot easily (and non-destructively) measure Young’s elastic modulus E . In addition, only a few violin E strings are made from a solid material (i.e. steel). Many modern-day violin E strings, and all other strings for bowed string instruments, are constructed from metal wound around a synthetic core (usually nylon or a nylon-like material) (Pickering 1985). I therefore need to estimate B from empirical measurements.

Unfortunately, peaks in the spectral domain do not only occur at the transverse modes of vibration, so I cannot simply pick the highest X peaks in the spectrum. In addition to transverse vibration, torsional vibrations (Woodhouse & Loach 1999) are present, and on some strings (notably viola D in Figure 3.5) energy can be found at the “ideal” frequencies predicted by (3.3). The origin of this energy is unclear and has been noted in the literature over the past decade (Conklin 1999) investigated “phantom partials” in piano tones occurring either at harmonic multiples, or at the sums of lower-frequency peaks; this is thought to occur due to string tension varying during transverse vibration and thereby causing longitudinal vibrations. (Woodhouse 2004) discussed similar behaviour in guitars, advancing the explanation of longitudinal vibrations but also suggesting that boundary

⁵Notation reminder: In this chapter, f_n and f_0 refers to frequencies, rather than modal forces.

conditions of the string against the bridge, nut, or fret could contribute to non-linear behaviour. (Penttilen et al. 2006) examined the guqin (a Chinese fretless plucked string instrument), finding peaks at both B and $B/4$. For low values of n and relatively low spectral resolution, a peak occurring as predicted by the stiff-string (3.4) will be indistinguishable from either the stiff-string (3.4) with B replaced by $B/4$ or a “phantom” peak at (3.3).

The focus of this dissertation is the synthesis and control of virtual string instruments, so I will make some simplifying assumptions. All of the above possibilities suggest that additional energy may be found at or above nf_0 . I will therefore assume that, in the first analysis window (8192 samples, or 85 ms), there will be a single peak close to the frequency predicted by (3.4) arising due to the transverse vibrations, and that any peak elsewhere is suspect and should be discarded. The lowest frequency of these suspect peaks occurs at nf_0 . The main concern is that when the mode numbers are high enough, the modal peak M will occur within the same frequency region as the non-linear peak

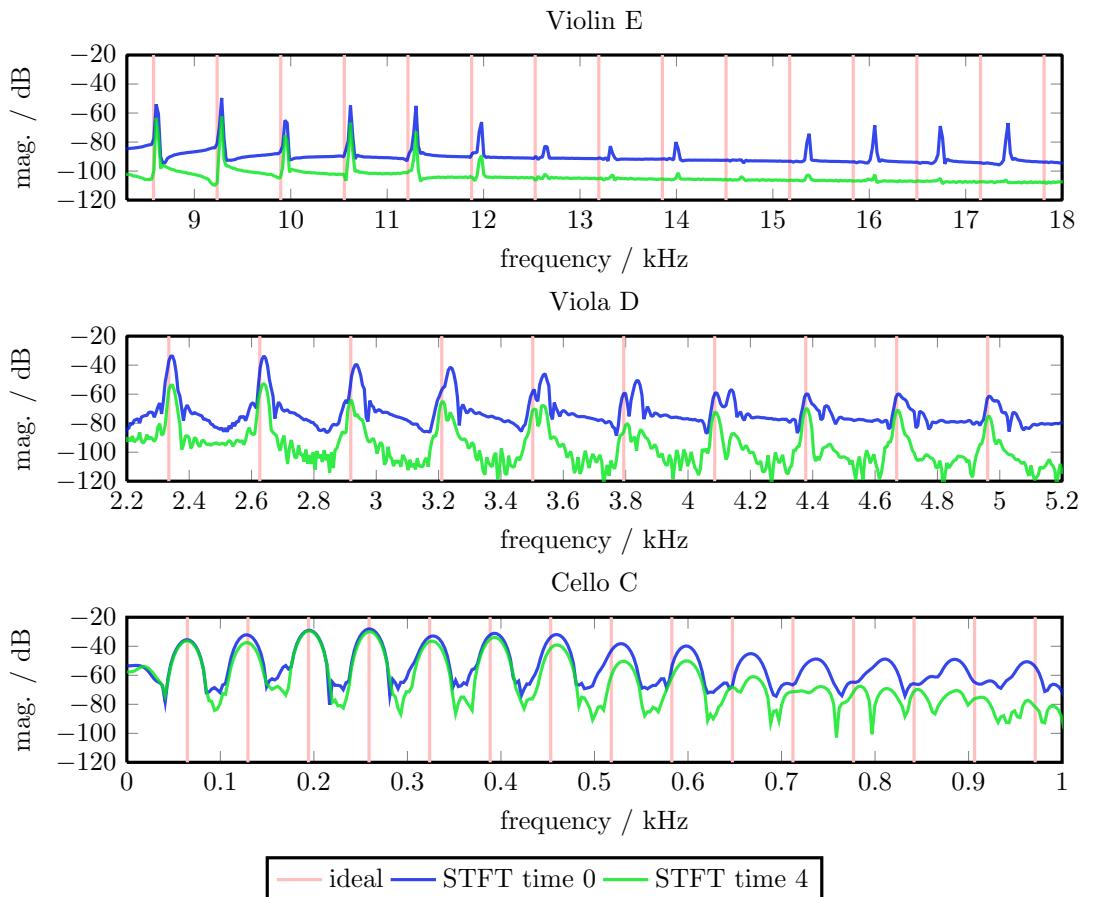


Figure 3.5: Non-linearities in the violin E, viola D, and cello C string. For the violin E string, note that the actual frequencies of the peaks almost matches the ideal frequencies at lower frequencies (10 kHz), but deviate sharply as the frequency increase. The viola D string shows lower-frequency peaks decaying in the expected manner, but the higher-frequency peaks appear to “split”. When such a “split peak” occurs, the higher of the two split peaks decays quickly (e.g., the peak at 4.7 kHz almost reaches the noise floor within 4 analysis windows), while the lower of the split peaks decays slowly. The cello C string does not demonstrate a great deal of “split peaks”, but the highest peak visible in the spectrum (the 14th node, at 0.96 kHz) almost touches the next “ideal” frequency (the 15th node, at 0.975 kHz).

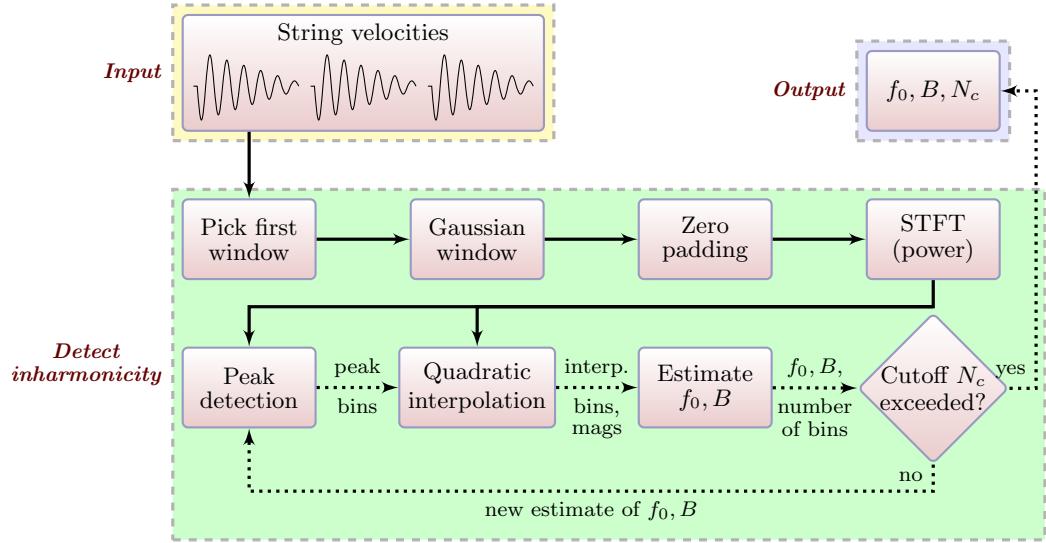


Figure 3.6: Detecting string inharmonicity. Solid lines indicate the signal or spectral information (only sent once); dotted lines indicate other variables (sent multiple times). Not shown in the diagram: Peaks whose magnitude is less than 20 dB above the noise $N_{top}(\omega)$ are discarded.

$M - 1$. I therefore define N_c as the cutoff for the number of partials in order to avoid such confusion. Specifically, I must find the lowest $N_c \in \mathbb{Z}^+$ such that

$$(N_c + 1)f_0 - S_b > N_c f_0 \sqrt{1 + BN_c^2} + S_a \quad (3.6)$$

where S_b and S_a are the allowable spread of frequencies below and above the exact frequency value predicted. The values $S_b = 0.1f_0$ and $S_a = 0.2f_0$ were found to work well with my data by manually inspecting the peaks detected.

The overall process for estimating f_0 and B is shown in Figure 3.6. The first analysis window is extracted from all plucks. The first 5 partials are extracted from the STFTs with an initial estimate that f_0 is the expected frequency of the string and $B = 10^{-4}$, then searching in the frequency region of $f_n - S_b$ to $f_n + S_a$ based on f_n calculated with string inharmonicity (3.4). Once the peaks were found, a least-squares fit was made to fit (3.7) to the detected frequencies. That is, given N detected frequencies where y_n is the interpolated frequency of partial n , the computer found parameters f_0 and B such that the following summation was minimized:

$$\sum_{n=4}^N \left(\frac{y_n}{n} - f_0 \sqrt{1 + Bn^2} \right)^2 \quad (3.7)$$

Partials 1, 2, and 3 were found to vary significantly, and were thus excluded from these fits. After the initial estimate of f_0 and B was made with partials 4 and 5, the STFT was re-examined in order to find the first 6 partials close to the frequencies predicted by (3.4) and the previous estimate of f_0 and B . This process is repeated, estimating a new f_0 and B then increasing the number of partials for which to search. Three examples of these fits are shown in Figure 3.7.

There are two conditions to determine the end of the process. First, all partials must be at least 20 dB higher than the noise $N_{top}(\omega)$; if no sufficiently large peak can be found within the frequency range, the process ends. Second, the process ends if the cutoff N_c (3.6) is reached.

name	f_0	$B \cdot 10^{-5}$	R^2	stiff-ideal mode cutoff N_c	notes
violin-E-I	659.7	4.90	1.00	30	
violin-E-II	658.8	5.13	0.99	30	
violin-E-III	660.3	5.06	1.00	30	
violin-E-IV	662.9	4.99	1.00	30	
violin-E-V	657.0	4.46	1.00	31	
violin-A-I	438.5	20.20	0.99	19	
violin-A-II	437.1	21.71	1.00	18	
violin-A-III	436.2	16.01	0.99	20	
violin-A-IV	442.1	25.19	0.99	17	
violin-A-V	439.5	22.59	1.00	18	
violin-D-I	291.0	22.59	0.94	18	(a)
violin-D-II	290.8	36.34	0.99	15	
violin-D-III	292.3	35.25	1.00	15	
violin-D-IV	295.2	36.70	1.00	15	
violin-D-V	292.2	45.96	0.99	14	
violin-G-I	194.4	25.94	0.96	17	
violin-G-II	194.6	19.48	0.97	19	
violin-G-III	194.9	26.21	0.93	17	(a)
violin-G-IV	197.6	21.07	0.95	18	
violin-G-V	195.5	37.25	0.99	15	
viola-A-I	439.0	5.96	0.99	28	
viola-A-II	444.8	9.66	1.00	24	
viola-D-I	291.7	13.41	0.98	21	
viola-D-II	292.8	6.20	0.98	28	(b)
viola-G-I	195.5	14.04	0.98	21	
viola-G-II	196.9	16.27	0.99	20	
viola-C-I	130.0	37.79	0.97	15	
viola-C-II	130.6	17.23	0.82	20	(a)
cello-A-I	219.2	6.28	0.98	27	
cello-A-II	220.0	5.69	0.99	28	
cello-A-III	218.6	6.02	0.99	28	
cello-D-I	146.1	16.75	0.99	20	
cello-D-II	146.5	24.91	0.97	17	
cello-D-III	145.9	12.79	0.98	21	
cello-G-I	97.1	19.99	0.99	18	
cello-G-II	97.2	23.40	0.98	17	
cello-G-III	97.4	21.74	0.97	18	
cello-C-I	64.7	62.64	0.98	12	
cello-C-II	65.1	58.20	0.99	13	
cello-C-III	64.5	53.52	0.97	13	

Table 3.3: Fundamental frequencies f_0 and inharmonicity B for measured strings. The variation of f_0 arises from instruments being tuned “by ear”.

(a) for these instruments, a few (< 10) “phantom” peaks were detected instead of the intended inharmonic peaks. However, manual inspection of the spectrum showed that the estimated B still produced adequate f_n estimates to capture the intended peaks.

(b) this instrument was old and not in normal playing condition; the viola D string was broken and had not been replaced. I replaced the D string with the A string (tuned down a fifth) as an “emergency fix” following normal violinist practice. The data from this string is still sufficient to enable casual synthesis, but should not be relied for any acoustics research on string behaviour.

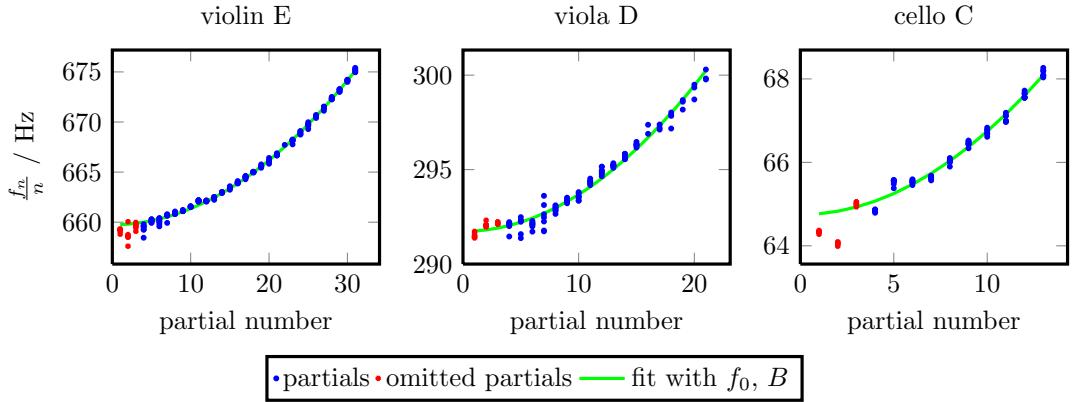


Figure 3.7: f_0 and B fits for the violin E, viola D, and cello C strings. The y axis shows the relative frequency of each partial divided by the number of that partial. An ideal, perfectly-flexible string would produce relative frequencies in a flat horizontal line.

The final results of the string inharmonicity detection are shown in Table 3.3. The table includes the coefficient of determination R^2 as a quick check of how closely the fit of f_0 and B matches the data, given \bar{y}_i as the mean of observed data y .

$$R^2 = 1 - \frac{\sum_i (y_i - f_i)^2}{\sum_i (y_i - \bar{y}_i)^2} \quad (3.8)$$

3.2.3 Detecting modal decays

There are two stages to the analysis of modal decays. First, the decay of each mode must be estimated from the plucks. In an ideal experiment with perfectly-controlled inputs, these plucks would yield exactly the same modal decays. In reality, some form of statistical analysis will be required to achieve a credible estimate. Second, a general formula which describes modal decays must be found. The stiff-ideal mode cutoff N_c in Table 3.3 varies from 12 to 31, yet my simulation will use 40 modes as discussed in Section 3.5.1. Since we cannot trust any modal decay rates above the N_c limit, I must extrapolate decay rates for the modes above N_c .

Tracking partials over time

In the previous section I estimated the fundamental frequency f_0 , inharmonicity constant B , and the highest mode number cutoff which can be safely detected N_c . The next step in detecting modal decays is to track these partials over time.

As with the estimation of f_0 and B , the range of frequencies examined for each mode will be $f_n - S_b$ to $f_n + S_b$, although when tracking partials over time I used $S_a = 0.05f_0$ and $S_b = 0.05f_0$ rather than the values I used when detecting f_0 and B . In order to reduce confusion from non-linear behaviour of the string, the strength of each partial is estimated from the spectral peak closest to the expected frequency f_n , rather than calculating the overall energy within a wide frequency band. Some examples are shown in Figure 3.8.

Before fitting those partials to exponential decays, I remove partials which are unlikely to have enough information to achieve a good fit. This process begins by estimating a per-partial noise floor. The final 10% of each partial is considered to be noise, and the top of the noise floor N_{par} is defined

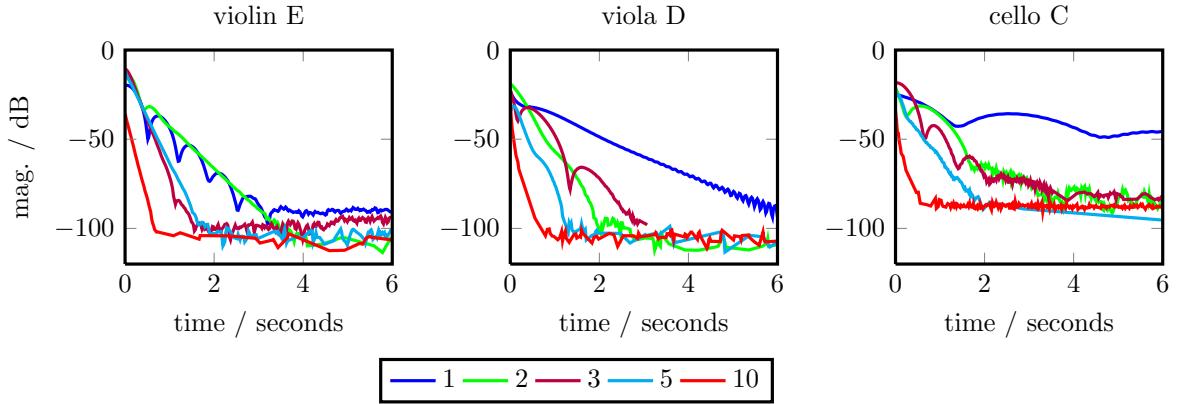


Figure 3.8: Selected partials of violin E, viola D, and cello C, one pluck. Legend indicates the number of the partial. The violin E string shows very good exponential decay with the exception of beating in the first partial. The viola D and cello C strings shows greater deviation from exponential decay,

as the 99th percentile of that data. Once N_{par} is calculated, the partial is rejected unless it satisfies the following two conditions:

- The maximum magnitude must be at least 30 dB above N_{par} .
- At least 10 samples must be at least 10 dB above N_{par} .

After removing unsuitable partials, the observed magnitudes y_t of remaining partials are least-squares fitted to

$$y_t = \alpha_2 + \alpha_1 e^{-\alpha_0 t} \quad (3.9)$$

with the constraints that α_0 , α_1 , and α_2 must all be greater than zero. The fit is performed to the logarithm of the measured data. The only important parameter for the physical modelling is the decay constant α_0 ; this is the estimate for r_n seen in (2.5).

name	candidate partials	maximum mag. $< N_{par} + 30$ dB	< 10 mag. $> N_{par} + 10$ dB	decays used	highest mode
violin-E-I	210	14	63	133	22
violin-E-II	210	41	60	109	17
violin-E-III	210	13	19	178	30
violin-E-IV	210	41	35	134	30
violin-E-V	217	29	66	122	22
violin-A-I	133	42	34	57	11
violin-A-II	126	35	35	56	9
violin-A-III	140	25	50	65	12
violin-A-IV	119	15	36	68	10
violin-A-V	126	24	51	51	9
violin-D-I	112	39	12	61	9
violin-D-II	105	25	28	52	10
violin-D-III	105	19	24	62	9
violin-D-IV	105	0	39	66	12
violin-D-V	98	26	15	57	13
violin-G-I	98	12	29	57	9
violin-G-II	112	7	32	73	11
violin-G-III	119	55	8	56	8
violin-G-IV	126	13	42	71	18
violin-G-V	105	5	36	64	12
viola-A-I	196	16	65	115	17
viola-A-II	168	14	70	84	12
viola-D-I	147	38	45	64	12
viola-D-II	147	12	48	87	17
viola-G-I	147	13	54	80	12
viola-G-II	140	7	37	96	14
viola-C-I	105	3	25	77	11
viola-C-II	126	2	30	94	18
cello-A-I	189	17	75	97	14
cello-A-II	196	10	58	128	22
cello-A-III	196	24	60	112	17
cello-D-I	140	14	39	87	15
cello-D-II	119	1	42	76	12
cello-D-III	147	1	49	97	19
cello-G-I	126	1	20	105	16
cello-G-II	119	0	0	119	17
cello-G-III	126	0	32	94	17
cello-C-I	84	3	7	74	12
cello-C-II	91	0	9	82	13
cello-C-III	91	0	6	85	13

Table 3.4: Removing suspicious partials from decay-fitting.

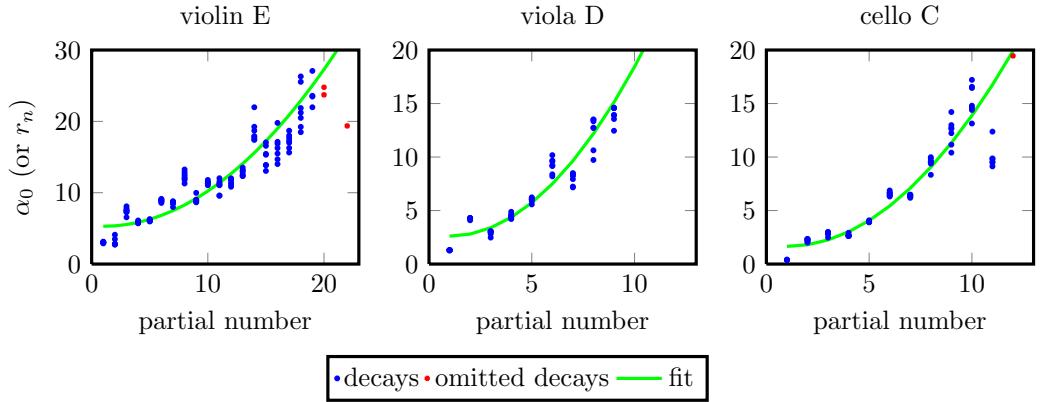


Figure 3.9: Detected and fitted decays for violin E, viola D, and cello C strings.

Extrapolating higher modal decays

Once estimates from each partial were established, I turned to the question of determining r_n for use in the simulation. For low modes which have extensive measured data, I set r_n to be the median value of α_0 for mode n . However, if less than half of the plucks yielded an estimate of α_0 for that mode (either due to partials being removed, or because n is above the cutoff N_c), I attempted to find a general formula linking the decay constant r_n to the mode number n for each string in order to extrapolate values for higher modes.

Two such formulae exist in the literature. (Adrien 1991) and (Demoucron 2008) used

$$r_n = B_1 + B_2(n - 1)^2 \quad (3.10)$$

while (Woodhouse 2004) used a much more complex formula involving loss factors $\eta = \frac{1}{Q} = \frac{2\alpha_0}{\omega}$. After the above substitution, his formula is

$$r_n = \frac{(nf_0\sqrt{1+Bn^2}) (T(\eta_F + \eta_A/(nf_0\sqrt{1+Bn^2})) + B\eta_B(n\pi/L)^2)}{2(T + B(n\pi/L)^2)} \quad (3.11)$$

where η_F , η_B , η_A are the fitted parameters representing the loss factors due to internal friction, bending, and air. Woodhouse remarked that his formula “should be understood as a combination of physically-based modelling and curve-fitting, since the physical mechanisms are not understood in sufficient detail to provide a fully convincing predictive model.” (Woodhouse 2004, p. 956)

Further study would be an interesting project, but as my main interest is in the *control* of the physical model rather than the physics itself, I instead made a choice between the existing two formulae for r_n . The chief difficulty with Woodhouse’s formula is that it relies on accurate estimates of tension T . The other constants — the frequency f_0 , inharmonicity B , and length L — are easy to estimate or measure directly. Tension can be estimated by pulling the string with a spring, but doing so will change the tension slightly, decreasing the accuracy of the estimate. I therefore turned to the simpler equation used by Demoucron, with the additional reassurance that his model produces audio which is adequate for my purpose.

Some examples of these fits are shown in Figure 3.9, while the fitted data for all instruments is in Table 3.3.

name	B_1	B_2	R^2	notes	r_1	r_2	r_3	r_4	r_5	r_{10}
violin-E-I	5.90	0.053	0.86		3.0	3.1	7.4	5.9	6.1	11.4
violin-E-II	7.62	0.080	0.84		2.9	3.9	7.8	6.1	8.6	15.3
violin-E-III	6.64	0.017	0.73		2.9	2.2	6.7	7.3	6.0	9.5
violin-E-IV	6.90	0.053	0.88		3.7	4.3	5.9	6.4	7.9	12.6
violin-E-V	5.30	0.063	0.88		2.8	2.6	6.6	4.1	7.7	11.7
violin-A-I	4.61	0.305	0.88		2.5	4.5	6.1	7.5	11.2	29.3
violin-A-II	3.80	0.324	0.85		1.7	4.8	5.5	8.6	11.0	30.0
violin-A-III	3.12	0.273	0.91		1.7	3.8	3.8	6.1	8.6	25.2
violin-A-IV	3.08	0.364	0.98		2.5	4.7	5.0	6.8	8.1	32.1
violin-A-V	3.24	0.435	0.96		2.1	4.8	4.8	8.0	10.5	38.5
violin-D-I	4.37	0.300	0.73		1.7	3.6	6.1	8.2	10.3	28.7
violin-D-II	3.44	0.290	0.75		1.2	4.1	3.5	8.5	8.8	26.9
violin-D-III	2.81	0.315	0.82		1.5	3.4	3.9	4.6	8.5	28.3
violin-D-IV	4.40	0.171	0.56		1.3	1.8	4.1	4.1	7.9	18.3
violin-D-V	0.78	0.554	0.92		1.0	2.1	3.6	4.5	9.1	45.6
violin-G-I	1.85	0.225	0.87		0.7	2.0	2.0	4.1	6.4	20.1
violin-G-II	1.04	0.211	0.86		0.6	1.0	2.8	3.6	4.3	20.0
violin-G-III	1.81	0.285	0.89		1.0	2.5	3.0	4.8	5.4	24.9
violin-G-IV	1.63	0.111	0.91		0.4	2.1	1.9	2.5	3.5	9.1
violin-G-V	1.39	0.256	0.76		0.6	1.5	1.6	3.8	5.8	22.1
viola-A-I	6.23	0.102	0.91		2.9	3.1	6.1	8.3	7.2	21.6
viola-A-II	4.04	0.141	0.93		2.3	2.3	5.3	4.7	5.4	15.7
viola-D-I	2.82	0.181	0.91		1.3	4.2	2.9	4.5	6.0	17.5
viola-D-II	4.02	0.090	0.84		0.9	2.9	5.1	4.5	6.7	12.1
viola-G-I	1.87	0.152	0.81		1.4	1.1	3.3	3.2	3.9	18.2
viola-G-II	0.16	0.157	0.94	(a)	0.9	0.7	1.4	1.7	3.2	10.4
viola-C-I	3.96	0.086	0.46		0.6	1.8	3.4	4.2	6.3	11.5
viola-C-II	1.84	0.081	0.85		0.6	2.1	1.3	3.1	3.5	8.1
cello-A-I	4.14	0.126	0.92		2.5	2.5	4.7	4.0	6.8	15.7
cello-A-II	2.38	0.061	0.95		1.6	3.3	2.8	2.9	3.1	6.7
cello-A-III	3.05	0.089	0.92		1.7	2.4	3.5	3.9	3.9	8.3
cello-D-I	1.84	0.196	0.98		0.8	1.8	3.8	4.1	4.9	17.8
cello-D-II	3.78	0.138	0.80		1.3	2.9	4.2	5.0	7.7	17.6
cello-D-III	1.75	0.093	0.88		1.8	1.5	3.4	2.4	2.9	10.2
cello-G-I	0.66	0.076	0.89		0.3	1.1	1.5	2.0	1.3	6.6
cello-G-II	1.05	0.090	0.93		0.6	2.4	1.2	1.8	2.6	7.6
cello-G-III	1.25	0.104	0.95		0.5	2.1	1.4	2.2	2.8	7.5
cello-C-I	2.33	0.125	0.80		0.4	2.2	2.8	2.7	4.0	15.3
cello-C-II	0.51	0.175	0.97		0.4	1.3	1.6	2.7	3.3	16.2
cello-C-III	1.64	0.090	0.91		0.4	1.4	1.3	2.1	3.1	11.3

Table 3.5: Fitted modal decays of measured strings. The fitted values B_1 and B_2 are used for modes which have fewer than 4 measured decays. The values given for r_n are drawn from direct measurements, with the exception of a few of the r_{10} values which were estimated from (3.10).

(a) although B_1 is suspiciously low, a manual examination of this fit looks plausible, especially given that on this string, all modes up to (and including) 14 are set as the median of the measured decays rather than estimated from B_1 and B_2 .

3.3 Instrument body impulse responses

Gathering data for the instrument body simulation is much easier than the string simulation, since the input to the simulation is a direct time-domain recording.

3.3.1 Experimental procedure and analysis techniques

I used the standard method of tapping the bridge and recording the audio output (Demoucron 2008). A newer method recommends exciting the instrument by damping the strings with a rubber mat, then pulling a thin piece of wire against a string until it breaks (Türckheim et al. 2010). A TASCAM LD-74 diaphragm condenser microphone was placed approximately 30 cm away from the bridge, and a small pendulum was held approximately 2 cm from the instrument bridge. The pendulum was held with a wooden frame⁶, while the displacement of the pendulum was standardized by holding it against a piece of cardboard serving as a guide. The pendulum was released and allowed to swing freely, hitting the bridge of each instrument with an approximately constant impulse. The pendulum then bounced off the bridge and hit the bridge again approximately 0.6 seconds later, but this second hit was not problematic as I only require 0.1 seconds of audio. The experimental setup is shown in Figure 3.10.

Audio was recorded at 44100 Hz with 24-bit samples, and at least seven taps were recorded from each instrument. In each recording, one good tap was manually chosen before proceeding to automatic processing. Each tap was downsampled by a factor of 2, then a high-pass Butterworth filter with cutoff 20 Hz was applied. The maximum absolute value in the resulting signal was found, and the tap was deemed to begin at the zero-crossing immediately before that maximum value, and last for 4096 samples. The length of the tap was further truncated to 1024 to match the required FFT length stated in Section 2.1.5. Since each instrument has different bridge and body responses, the recorded audio signals have different maximum and minimum amplitudes. These signals are deliberately not normalized; if one real-life instrument sounds twice as loud as another one when given the same string signal, then the physical model should reproduce that behaviour. If two musicians want to produce music that is perceived to be the same volume, then they (or the feedback control) must alter their bowing parameters.

⁶The wooden frame was constructed by Dr. Paul Percival from Simon Fraser University, Canada.



Figure 3.10: Photos of the pendulum for tapping the bridge. Left: instrument, pendulum, and microphone. Right: close-up of the instrument, with the pendulum (black ball) held against the guide with a wooden stick. When the wooden stick is removed, the pendulum falls to the left, travelling ≈ 2 cm before hitting the instrument bridge.

3.3.2 Instrument body impulse responses

A time-domain comparison of a violin and a cello impulse response is shown in Figure 3.11, while a comparison of the spectrums of all impulses responses is shown in Figure 3.12.

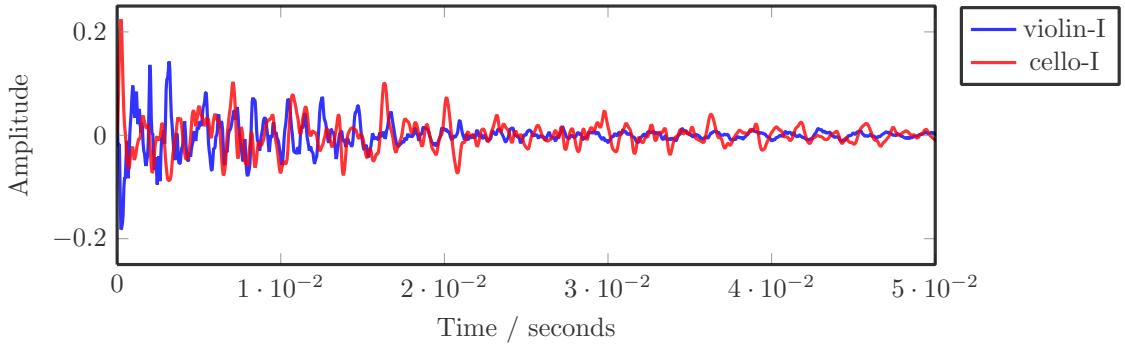


Figure 3.11: Time-domain impulse response of a violin and a cello with a standard tap. Note that the violin response decays faster than the cello response.

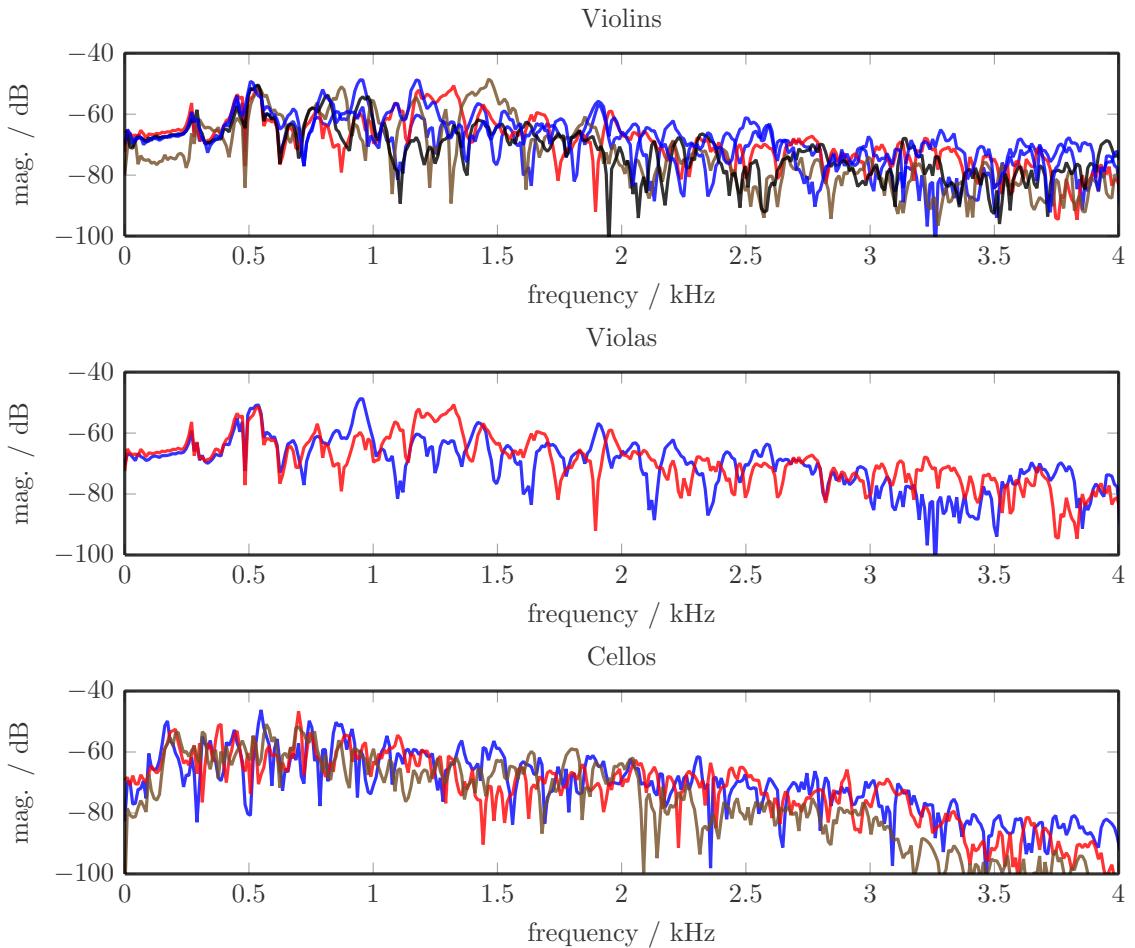


Figure 3.12: Frequency impulse responses of all instrument bodies. Note that most instruments agree quite well for the lower frequencies.

3.4 Estimating remaining constants

Some physical constants are easy to measure, such as the string length L , but other constants are difficult to measure non-destructively. For example, although the vibrating portion of a violin string has nearly constant linear density, the ends of the string have additional material; measuring the mass of the entire string would over-estimate the mass of the vibrating portion of the string.

In Section 3.1 I measured string length L and diameter d , while in Section 3.2 I measured string modal decays r_n . This leaves three string constants to estimate: tension T , linear density ρ_L , and Young's elastic modulus E . In addition, there are three bow-string friction constants needed: coefficients of static friction μ_s , dynamic friction μ_d , and the slope of the hyperbolic curve μ_c . The only constants relating to the instrument are the impulse responses measured in Section 3.3.2.

3.4.1 Bounds of physical constants in the literature

Table 3.6 shows the range of physical constants for string T , ρ_L , and E . These ranges will be used to estimate values to match the f_0 and B of each string measured in Section 3.2.2. Table 3.7 shows the range for bowing friction constants.

instrument	ρ_L		E		T	
	min	max	min	max	min	max
violin E	$3.80E-4$	$4.80E-4$	$4.5E+9$	$2.2E+11$	71.4	90.7
violin A	$5.80E-4$	$7.50E-4$	$4.5E+9$	$2.2E+11$	48.3	62.7
violin D	$9.20E-4$	$1.63E-3$	$4.5E+9$	$2.2E+11$	34.3	60.6
violin G	$2.12E-3$	$3.09E-3$	$4.5E+9$	$2.2E+11$	35.0	51.1
viola A	$5.60E-4$	$9.20E-4$	$4.5E+9$	$2.2E+11$	60.6	100.2
viola D	$9.80E-4$	$1.25E-3$	$4.5E+9$	$2.2E+11$	47.6	60.7
viola G	$2.20E-3$	$2.81E-3$	$4.5E+9$	$2.2E+11$	47.6	60.7
viola C	$4.95E-3$	$6.31E-3$	$4.5E+9$	$2.2E+11$	47.6	60.7
cello A	$1.50E-3$	$1.92E-3$	$4.5E+9$	$2.2E+11$	138.3	177.2
cello D	$2.94E-3$	$3.57E-3$	$4.5E+9$	$2.2E+11$	121.0	146.9
cello G	$6.38E-3$	$7.56E-3$	$4.5E+9$	$2.2E+11$	116.8	138.3
cello C	$1.43E-2$	$1.70E-2$	$4.5E+9$	$2.2E+11$	116.7	138.3

Table 3.6: Bounds of string constants used in the bowed-string algorithm. Tension T and linear density ρ_L came directly from (Rossing 2010, p. 286). Young's elastic modulus E came from (Jansson 2002, table 4.7), allowing strings to be any material from nylon to solid steel.

instrument	μ_s		μ_d		μ_c	
	min	max	min	max	min	max
violin	0.60	0.90	0.20	0.40	0.05	0.30
viola	0.70	1.00	0.20	0.50	0.05	0.30
cello	0.80	1.20	0.30	0.50	0.05	0.30

Table 3.7: Bounds of bowing constants used in the bowed-string algorithm. These values come from examining constants used in (McIntyre et al. 1983, Smith & Woodhouse 2000, Serafin 2004, Inácio et al. 2008). Friction mainly arises due to rosin, a sticky substance scraped onto the bow hair by the musician which gradually wears off. Rosin for violins is lighter in color, less sticky, and often described as "harder". Rosin for cellos and especially double basses is dark, stickier, and "softer".

3.4.2 Fitting unknown string constants to measured constants

Selecting the bow-string friction constants begins by uniformly randomly selecting a value from the ranges in Table 3.7. This value represents the inherent friction of the bow and rosin. The actual friction values are unlikely to be independent — a rosin which produces a high μ_s will likely have a high μ_d as well — but this does not substantially change the simulation. Once a value is selected for each instrument’s bow, the actual friction constants are multiplied by a further uniform random value between 0.95 and 1.05. This represents the variation in friction between the bow and each string; real-world violin strings have rosin residue from being bowed in the past. Although many musicians attempt to remove this residue from the string by wiping them with a cloth after playing the instrument, the residue cannot be removed entirely without using rubbing alcohol or a similar cleaning agent.

Unfortunately the bounds on string and bow constants given in Section 3.4.1 are listed as independent values (e.g., there is no published correlation between string diameter d and linear density ρ_L). In real life, we should expect some dependence between the string constants. However, I made the unrealistic (yet adequate for the resulting sound) assumption that physical constants are independent.

To estimate the unknown string constants T , ρ_L , and E , I found a least-squares fit to match those with the measured f_0 and B for each string. In addition, to accommodate measurement error in L and d , I treated those two values as variables which fall within 1% and 10% of their measured values, respectively. In order to keep the constants within the required bounds, I defined a $W(a, b^{\min}, b^{\max})$ function which adds a penalty if a is outside of the bounds $b^{\min} \leq a \leq b^{\max}$. Concretely, the problem is to find values for $[\rho_L, T, E, L, d]$ which minimize $\sum y_i^2$ in

$$\begin{aligned} W &= \max\left(\frac{b^{\min} - a}{b^{\min}}, 0\right) + \max\left(\frac{a - b^{\max}}{b^{\max}}, 0\right) \\ y &= \left[f_0 - \sqrt{\frac{T}{\rho_L} \left(\frac{n\pi}{L}\right)^2 + \frac{EI}{\rho_L} \left(\frac{n\pi}{L}\right)^4}, \quad B - \frac{E\pi^3 d^4}{64L^2 T}, \quad W(\rho_L, \rho_L^{\min}, \rho_L^{\max}), \right. \\ &\quad \left. W(T, T^{\min}, T^{\max}), \quad W(E, E^{\min}, E^{\max}), \quad W(L, L^{\min}, L^{\max}), \quad W(d, d^{\min}, d^{\max}) \right] \end{aligned} \quad (3.12)$$

The above equations assume that the string is a uniform beam, i.e. not a wound string. As previously discussed, this is a false assumption for almost all strings, but this assumption is fundamental in the physical modelling equations in Chapter 2. The real-world measured values of string constants produce estimates of B which are much too low. In order to achieve virtual strings whose inharmonicity factor B , and therefore modal frequencies w_n matches the measurements, I have allowed the Young’s modulus E to be considerably larger. The core of wound strings is nylon, so we should expect E to be close to 4 or 5 GPa. However, I have allowed E to be as high as 220 GPa, the value for solid steel strings found occasionally on the violin E string.

The string tension T falls somewhere between a physical constant and a musician-controlled value. It generally does not change significantly⁷ during normal playing, however at the beginning of each playing session the musician will adjust the string tension to ensure that the open strings produce the desired pitches. The initial estimate of T comes from (3.12), but it will be automatically adjusted to ensure that the string’s f_0 matches the desired frequency as discussed in Section 3.5.4.

⁷If the ambient temperature changes, the string will expand or contract, producing audible variation in pitch. This is unfortunately common when performing on a stage with high-power stage lighting; musicians will “re-tune” (i.e. adjust the string tensions) several times during a concert to mitigate this problem.

3.5 Constants selected from simulations

This section discusses simulations which were performed to choose addition constants. In particular, in the first portion of this section describes the choice of the number of modes (N), string sampling rates (f_s), finger damping and plucking constants (K_f , R_f , K_p , R_p), tuning the string tension (T), and the cut-off for a barely vibrating string (A_{\min}).

The simulation described in 2.1.1 contains two major simplifications: The model treats external forces F_i as constant throughout the time interval $dt = \frac{1}{f_s}$, and the model only represents the string with a finite number of modes N . This introduces a trade-off of speed vs. accuracy. A higher sampling rate will reduce the time interval dt , thereby improving the accuracy of the estimate of a constant external forces. Similarly, increasing N would bring the model closer to an infinitely large N , which is the mathematical assumption behind the modal decomposition. However, increasing f_s or N will require more calculations, which will slow down the simulation. For computational efficiency, N will be truncated to a finite number. This introduces a trade-off between mathematical accuracy and the number of calculations required. In terms of computation complexity, the algorithm has a time complexity of $O(N \cdot dt)$.

3.5.1 Number of modes

Plucking an open string imparts a certain amount of energy to each mode in the plucking phase, but during the release phase each mode decays independently. The importance of upper modes can be estimated from the modal decays predicted in Section 3.2.3, so I will not include simulations of open-string plucks in this analysis. A more interesting question is how the system behaves when a fingered string is plucked. As discussed in Section 2.1.2, the finger forces create the desired pitch by distributing energy between modes. This is a much better test of the effect of the limited number of modes for plucking.

Due to the way that external forces distribute energy between modes, a “non-existent” mode (i.e. mode numbers above N) can be considered to have an infinitely strong damping factor α . Therefore, the higher the damping of upper modes of a string, the less inaccuracies are introduced due to a finite N . Bowing a string results in a continual (although varied) addition of energy to the system. This additional energy is very important for the upper modes; the upper modes play a much more important role in bowing than they do in plucking.

I compared the two “extreme” strings: The cello C string (65 Hz), and violin E string (660 Hz). Upper modes of the cello C string are heavily damped, so it is expected that low values of N will be fairly accurate for plucked strings. By contrast, the upper modes of the violin E string are relatively lightly damped, so low values of N will be inaccurate for plucked strings. I tested $N = [32, 40, 48, 64]$, and set $f_s = 96$ kHz to avoid any problems of frequency aliasing. For computational efficiency the number of modes should be a multiple of 4 or even 8, as discussed in Section 4.2.2. Figure 3.13 confirms these intuitions. The bowed examples show a greater effect of changing N ; there is good agreement of low-frequency spectral peaks, but after the maximum frequency ω_N is reached the difference between the numbers of modes is clearly visible (and audible).

After considering the cello C examples, I chose to use $N = 40$ as being sufficient for my purposes. This retains the spectral peaks up to approximately 3 kHz; as we saw in the cello body impulse responses in Figure 3.12, the cello bodies vibrate much less for frequencies above 3 kHz. The sensitivity

of human hearing does not fall drastically until 5 kHz or 6 kHz (Suzuki & Takeshima 2004), so this is not a perfect situation — however, it should be noted that the cello G string will have a maximum modal frequency between 4 kHz and 4.5 kHz, while the cello D string will reach more than 6 kHz.

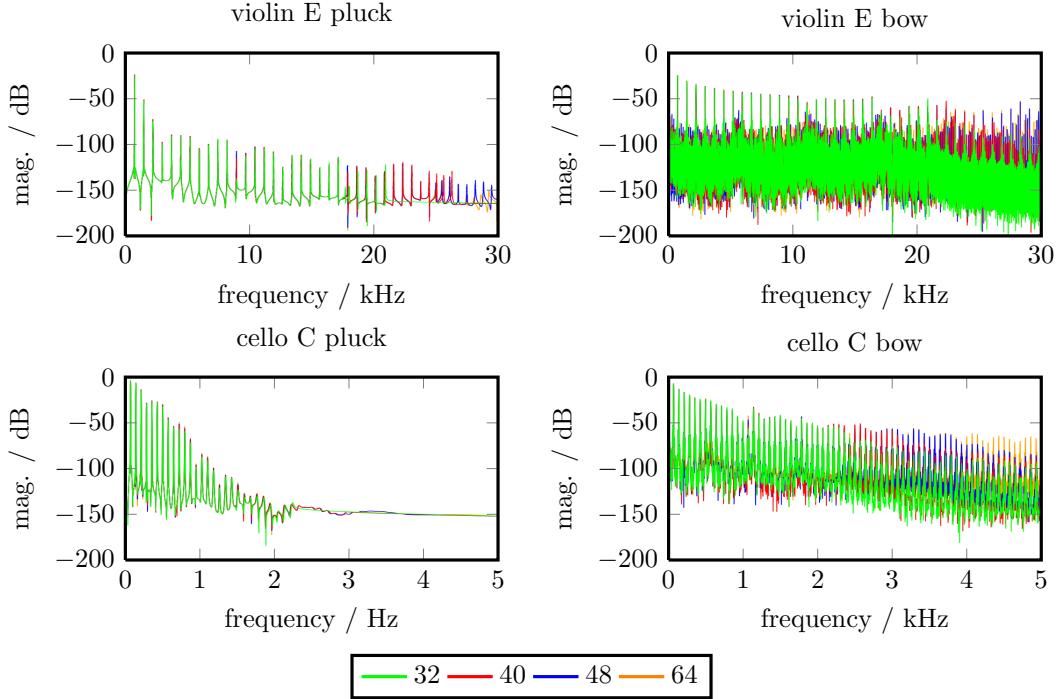


Figure 3.13: Different modes N for the plucked and bowed violin E and cello C strings. For violin E, the string was fingered at $x_f = 0.891L$, plucked at $x_p = 0.2L$, and bowed at $x_b = 0.12L$, $F_b = 0.15$, $v_b = 1.5$. For cello C, the string was fingered at $x_f = 0.891L$, plucked at $x_p = 0.2L$, and bowed at $x_b = 0.12L$, $F_b = 1.5$, $v_b = 1.5$.

Note that there is virtually no difference between modes 32 and 64 for the plucked cello example. By contrast, there is a clear difference in both the bowed examples: The peaks for $N = 32$ decrease in magnitude after 2.5 kHz.

Audio 3.2: Violin E plucks with varying number of modes N .

```
http://percival-music.ca/dissertation/a.3.2.ns-32-finger-violin-e.wav
http://percival-music.ca/dissertation/a.3.2.ns-40-finger-violin-e.wav
http://percival-music.ca/dissertation/a.3.2.ns-48-finger-violin-e.wav
http://percival-music.ca/dissertation/a.3.2.ns-64-finger-violin-e.wav
```

Audio 3.3: Violin E bowed notes with varying number of modes N .

```
http://percival-music.ca/dissertation/a.3.3.ns-32-bow-violin-e.wav
http://percival-music.ca/dissertation/a.3.3.ns-40-bow-violin-e.wav
http://percival-music.ca/dissertation/a.3.3.ns-48-bow-violin-e.wav
http://percival-music.ca/dissertation/a.3.3.ns-64-bow-violin-e.wav
```

Audio 3.4: Cello C plucks with varying number of modes N .

```
http://percival-music.ca/dissertation/a.3.4.ns-32-finger-cello-c.wav
http://percival-music.ca/dissertation/a.3.4.ns-40-finger-cello-c.wav
http://percival-music.ca/dissertation/a.3.4.ns-48-finger-cello-c.wav
http://percival-music.ca/dissertation/a.3.4.ns-64-finger-cello-c.wav
```

Audio 3.5: Cello C bowed notes with varying number of modes N .

```
http://percival-music.ca/dissertation/a.3.5.ns-32-bow-cello-c.wav
http://percival-music.ca/dissertation/a.3.5.ns-40-bow-cello-c.wav
http://percival-music.ca/dissertation/a.3.5.ns-48-bow-cello-c.wav
http://percival-music.ca/dissertation/a.3.5.ns-64-bow-cello-c.wav
```

Still, researchers interested in “fuller” cello C string would be advised to increase N , at the cost of a slower simulation.

A similar analysis was performed by (Demoucron 2008), who concluded that there was very little difference for N above 30 for the violin D string. However, it should be noted that his modal decay constants were much higher than the ones I measured in Section 3.2.3.

3.5.2 String sampling rates

Recall that the simulation of the instrument body happens at 22050 Hz and that the string sampling rate is a (discrete) multiplier M_f of the instrument sampling rate, as discussed in Section 2.1.5. Having selected the number of modes N , it would be simple to select a multiplier for each string which kept the highest modal frequency ω_n below the Nyquist frequency. However, as discussed in Section 2.3.2, even sampling rates well above the Nyquist frequency can still result in the bow skipping over the sticking-state with constant bowing parameters. To avoid those problems, the string sampling rate may need to be higher than the Nyquist frequency.

The suitability of different frequency multipliers M_f is evaluated using the normalized spectral centroid difference (SCN) as defined in Section 2.3.1. For each M_f , the string is bowed at two different bow velocities ($v_b = 0.1$ m/s, $v_b = 0.4$ m/s), multiple bow forces (20 forces linearly spaced from $F_b = 0.01$ N to $F_b = 3.0$ N), and multiple bow positions (73 values of x_b from $x_b = \frac{1}{17}L$ to

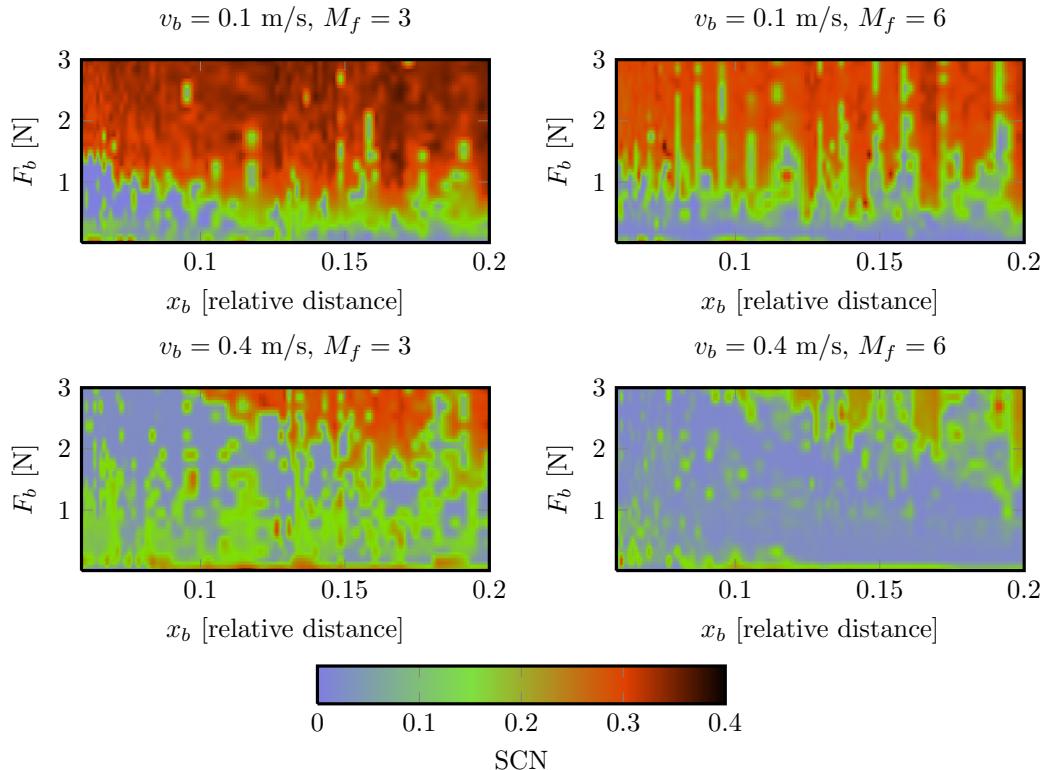


Figure 3.14: Testing parameters for frequency multipliers M_f . Violin E string, evaluated at bow velocity $v_b = 0.1$ and $v_b = 0.4$, at sample rates 66150 Hz and 132300 Hz. Lower values of SCN are good, as they indicate that more energy is concentrated at the desired frequency, and thus the string is vibrating at closer to Helmholtz motion.

$x_b = \frac{1}{5}L$, distributed between modal positions). An example is shown in Figure 3.14.

Once all simulations are performed for a particular frequency multiplier M_f , the median and mean SCN values for each of the two bow velocities are computed, giving four values which represent the overall quality of audio. Four strings of all three instruments are evaluated, and shown in Figure 3.15.

Just as in the case of choosing the number of modes N , there is no firm rule for choosing between the quality of audio (in this case, lower SCN) and the speed of simulation. In all cases, increasing the frequency multiplier M_f produces better audio, but in most cases there are only modest gains after an initial improvement. The values of M_f which I chose are shown in Table 3.8.

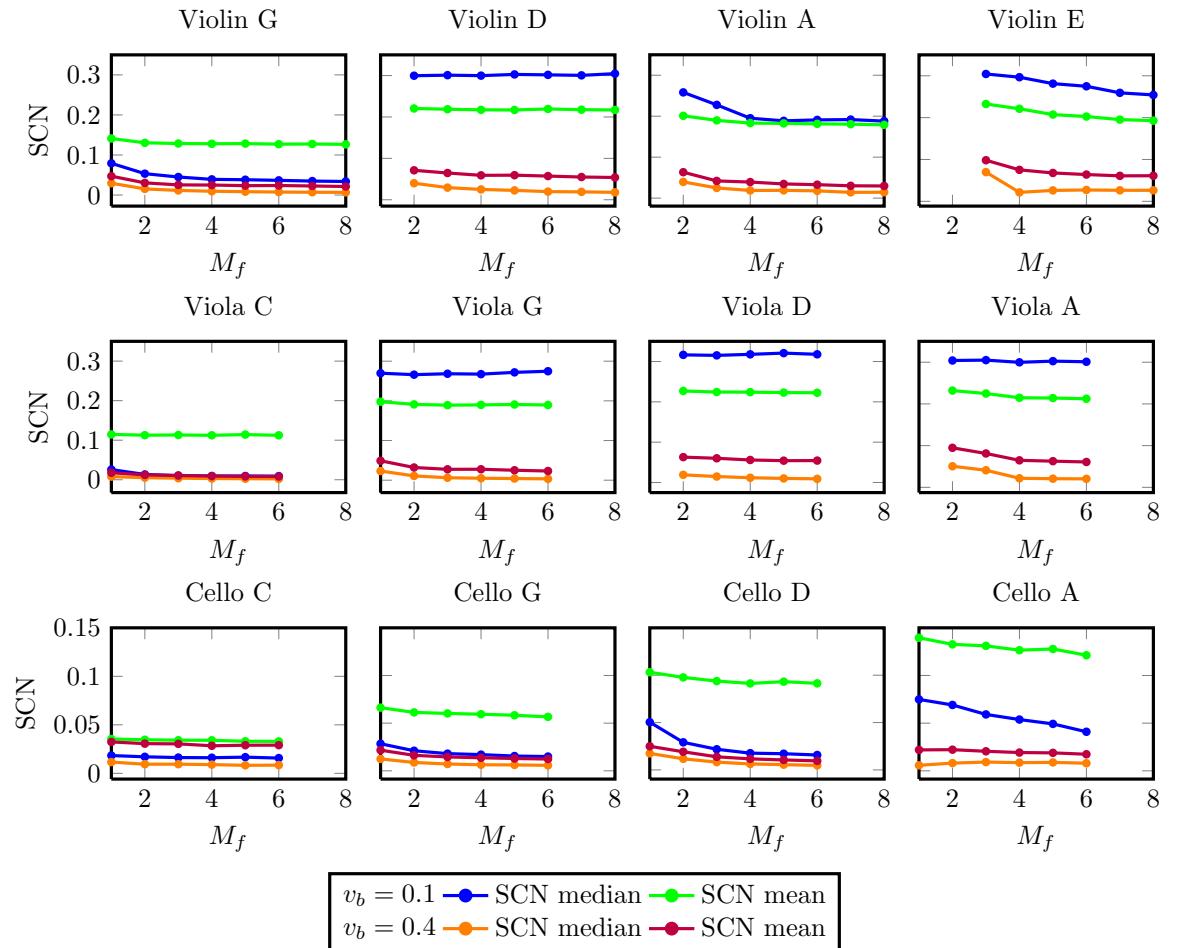


Figure 3.15: Audio quality resulting from frequency multipliers M_f , all strings. The viola and cello strings were only tested up to $M_f = 6$, and strings whose highest modal frequency ω_{40} was less than the Nyquist rate were not tested. Note that the same range of forces were used for all strings, so the lower values seen for the G and C strings is expected.

String	f_s multiplier	String	f_s multiplier	String	f_s multiplier
Violin E	4	Viola A	4	Cello A	2
Violin A	4	Viola D	2	Cello D	2
Violin D	2	Viola G	2	Cello G	2
Violin G	2	Viola C	2	Cello C	1

Table 3.8: String frequency multipliers.

3.5.3 Finger damping and plucking constants

This section discusses the left-hand finger and right-hand plucking finger spring constants K and damping constants R .

Finger and pluck spring constants K_f and K_p

The finger and pluck spring constants were chosen such that the string would move quickly when being plucked (requiring a high K_p) yet the finger position would not change significantly (requiring a high K_f). After examining various graphs such as Figure 2.4 and listening to the resulting audio, $K_f = 10^5$ and $K_p = 10^4$ was chosen.

Finger damping factor R_f

To get a rough estimate of the finger damping factor R_f , I measured plucks on a real violin G string to compare the open G string (no finger) with first finger. The RMS of the decay portion of each pluck is shown in Figure 3.16.

The recorded pluck decays for both the fingered and open-string plucks show non-linear decay. For example, at 1.0 seconds, the recorded magnitudes of the open-string plucks were significantly less than the simulated open-string pluck magnitudes. However, by 3.0 seconds the open-string plucks' magnitudes were roughly equal to the simulated values. Although the open-string simulation roughly matched the recorded values, the fingered string showed significant deviation from the simulated values depending on R_f . I tested $R_f = [0, 10, 20, 30, 100]$; the simulation with $R_f = 0$ exhibited slower decay than the open string. Of the values tested, $R_f = 30$ provided the closest match to the recorded plucks.

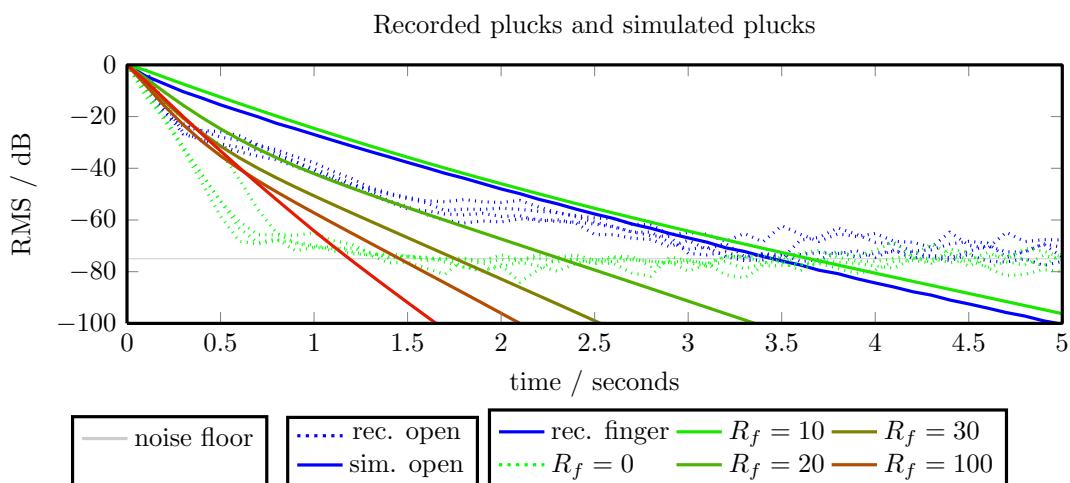


Figure 3.16: RMS decay for violin G string with open plucks and fingered plucks (recorded and simulated). Unlike previous simulations of the violin G string, this simulation includes the effects of the violin body. Each type of recorded pluck (open string, fingered first position) was repeated 5 times. Simulations were performed with $x_f = 0.891L$, $x_p = 0.2L$; similar positions were used for the recorded plucks. Decibels (dB) are normalized on a per-file basis to indicate the decay relative to that file's highest RMS.

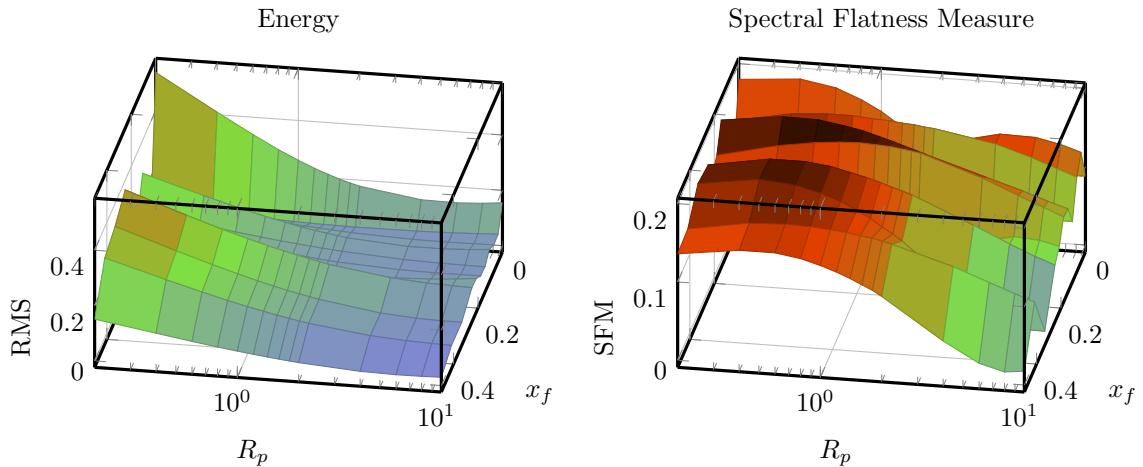


Figure 3.17: RMS and SFM during the “pull” portion of a pluck. Two plucks were performed at $x_p = 0.2L$ on the violin G string, while x_f and R_p were varied.

Audio 3.6: Examples of two plucks, $R_p = [0.2, 1.0, 10.0]$

<http://percival-music.ca/dissertation/a.3.6.pluck-Rp-0.2.wav>
<http://percival-music.ca/dissertation/a.3.6.pluck-Rp-1.0.wav>
<http://percival-music.ca/dissertation/a.3.6.pluck-Rp-10.0.wav>

Plucking damping factor R_p

As discussed in Section 2.2.1, plucking an already-vibrating string can result in unpleasant “clicking” sounds during the “pull” portion of the pluck. These sounds occur to some degree with real-life plucks, but this unwanted sound is much worse in the simulation with no plucking damping ($R_p = 0$). I therefore searched for a value of R_p which would minimize the unwanted vibrations during the plucking phase. To choose a value for the plucking damping factor R_p , I simulated two plucks with varying finger position x_f and damping factors as was done in Figure 2.13. The “pulling” portion of the second pluck was filtered with a high-pass fifth-order Butterworth filter with pass-band 40 Hz and stop-band 20 Hz, in order to remove the frequency of the finger pulling the string. The resulting signal of the “pulling” portion was analyzed in two ways. First, the overall RMS energy was computed; this provides a first approximation of how “loud” the sound will be perceived. Second, the spectral flatness measure (SFM) between 50 Hz and 5000 Hz was computed; the SFM measures how “tone-like” the sound will be, with a value of 1.0 representing white noise and lower values being pure tones.

These values are shown in Figure 3.17. After listening to numerous simulations, I decided that maximizing the SFM and minimizing the energy produced the least noticeable “pulling noise”. Lacking a reliable method of weighting these two factors, I chose $R_p = 1$ as a compromise. The actual RMS and SFM varied quite a bit depending on the finger position x_f , with “strongly rational” positions producing lower RMS and higher SFM than other positions.

3.5.4 Tuning string tension

We have no guarantee our model is mechanically accurate, but for musical use it is important that the open strings produce the desired pitches. To accommodate any potential inaccuracies, I do not use the estimated tension directly. Rather, I adjust the string’s tension to produce the correct pitch, just as is done by real musicians at the beginning of each playing session.

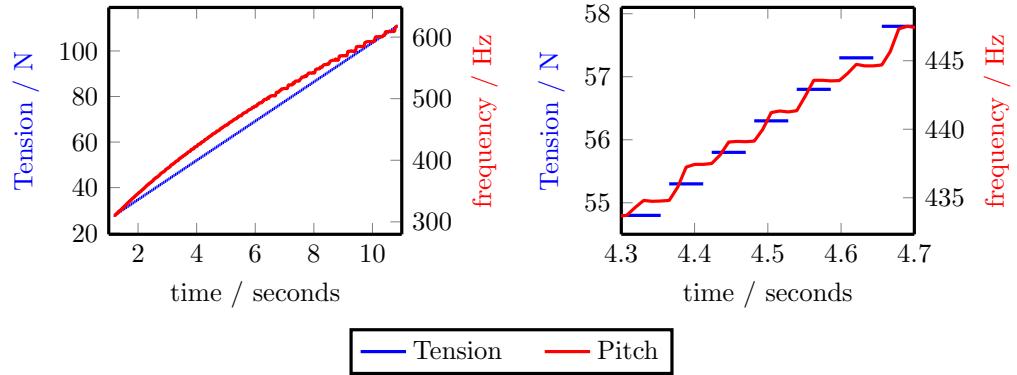


Figure 3.18: Relationship of tension to pitch, violin A string. The string was bowed with a constant $x_0 = 0.1$, $F_b = 1.0$, $v_b = 0.4$, with the initial string tension T drastically lowered. The string was bowed for 1 second to let the pitch “settle”, then the tension was gradually increased with a step function to show the lag between tension and pitch.

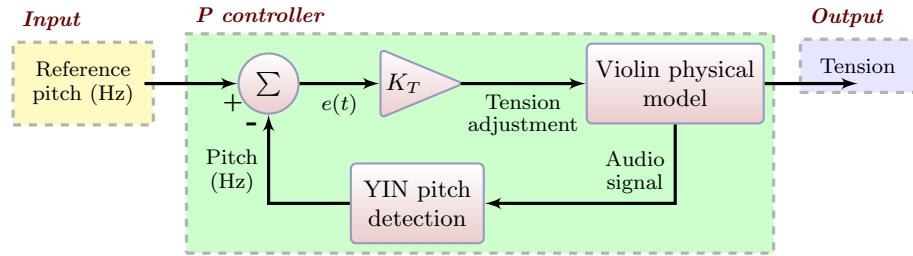


Figure 3.19: Pitch-tuning tension control loop. For demonstration purposes, the audio examples were generated after reducing the string tension T to 90% of the initial estimate found in Section 3.4.2. Audio 3.7: Pitch-tuning tension control loop in action.

```

http://percival-music.ca/dissertation/a.3.7.tune-violin-g.wav
http://percival-music.ca/dissertation/a.3.7.tune-violin-d.wav
http://percival-music.ca/dissertation/a.3.7.tune-violin-a.wav
http://percival-music.ca/dissertation/a.3.7.tune-violin-e.wav

```

Although we can find the frequency of the lowest mode by combining (2.5) and (2.7),

$$\omega_1 = \sqrt{\frac{T}{\rho_L} \left(\frac{\pi}{L}\right)^2 + \frac{EI}{\rho_L} \left(\frac{\pi}{L}\right)^4 - r_1^2} \quad (3.13)$$

the actual frequency of the produced audio does not follow this precisely. As discussed in Section 2.1.2, fingered notes achieve their pitch due to energy being distributed between modes at the finger positions x_0 and x_1 . Furthermore, even the pitch of an open string will vary based on bowing parameters — a string has a “rounded corner”, and the degree of rounding produces a pitch flattening effect which has been observed both theoretically and experimentally (McIntyre et al. 1983, Demoucron 2008, Schoonderwaldt 2009). Although ω_n has no precise relationship with the perceptual pitch, it is reasonable to assume that provided other parameters are held constant, the pitch will increase if ω_n increases as shown in (3.13).

To test the system as a whole, I performed pitch detection using an extension to the YIN algorithm (Brossier 2006, de Cheveigné & Kawahara 2002). The relationship of tension to pitch is shown in Figure 3.18. Note that the system behaves fairly simply when there are small variations in tension and adequate time is given to allow the system to adjust.

Given the simplicity of adjusting tension in the range we care about, and only needing to adjust the tension once (after the other physical constants are set) with no time constraints, I used a proportional controller Figure 3.19. The string was bowed at $x_b = 0.12L$ with an acceleration of $v_a = 0.2 \text{ ms}^{-2}$ up to a maximum velocity of $v_b = 0.3 \text{ m/s}$, while the force was set to a portion of the maximum Schelleng bow force discussed in Section 1.2.1,

$$F_b = K_s v_b \frac{2\sqrt{T\rho_L}}{\frac{x_b}{L}(\mu_s - \mu_d)} \quad (3.14)$$

The K_s factor reduces the bow force from the maximum; I experimentally found that $K_s = 0.2$ produced reasonable bow-strokes. The string was bowed from rest for 1.0 seconds, then the control loop was activated. I set $K_T = 0.1$ experimentally. The loop adjusted the tension until the relative error $\frac{f^r - f}{f^r} < 0.001$, where f is the median of the detected frequencies from the past 5 analysis windows, and f^r is the reference pitch.

3.5.5 Calculating cut-off displacement and velocity

When a simulated string is freely oscillating (i.e. a pluck has been released or the bow is touching that string), the decay is very close to exponential. The output of the instrument body will end abruptly when the signal reaches reaches the smallest positive value of 16-bit integers (i.e. 1), but the internal floating-point values for a_n and \dot{a}_n will continue calculating until they reach the smallest positive value of 32-bit floating values (i.e. 2^{-126} not including denormalized numbers). These values are

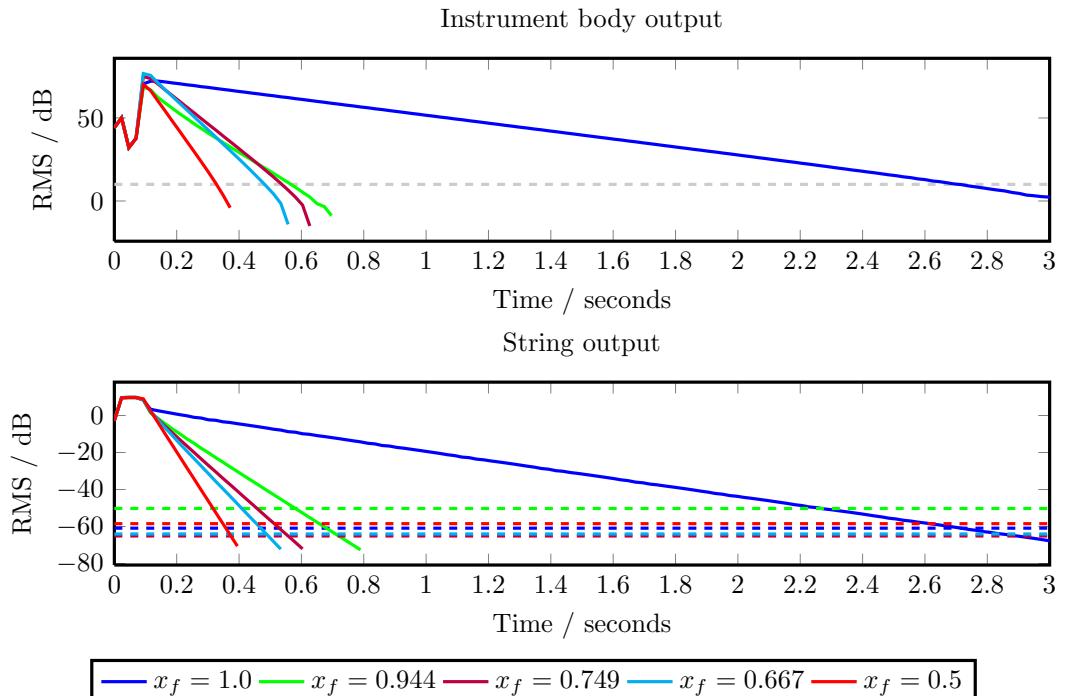


Figure 3.20: RMS output for instrument body and violin E string. The finger positions x_f shown here correspond to the musical values of open E, F, A, B, and E (an octave above the open string); the full experiment used all semitones between E and high E. The dotted horizontal line in the instrument output is the cutoff at 10 dB, while the dotted horizontal lines in the string output correspond to the A_{mag} at the time which that finger's output meets the instrument decay cutoff.

String	Pluck force	String	Pluck force	String	Pluck force
Violin E	0.5	Viola A	0.5	Cello A	0.6
Violin A	1.0	Viola D	1.0	Cello D	0.8
Violin D	1.0	Viola G	1.0	Cello G	0.9
Violin G	1.5	Viola C	1.75	Cello C	2.0

Table 3.9: Equal-loudness plucking force. These values were empirically selected by ear.

slightly misleading, as the instrument body filter is scaled to ensure that the string output produces reasonable values within the range of 16-bit integers. However, even with that scaling, 32-bit floats can still represent much smaller values than 16-bit integers. If the simulation continued until a_n and \dot{a}_n fell below the minimum positive values for floats, a great deal of computation would be wasted to produce an instrument output of 0. To avoid this useless computation, I turn off the simulation when $A_{\text{mag}} = \sum_i A[i]^2$ falls below A_{min} , as mentioned in Section 2.1.4.

Figure 3.20 shows the process of selecting A_{min} . I selected 10 dB as the threshold of significance for instrument output. I then simulated 13 plucks for each string; one for each semitone between the open string and its octave. The amount of plucking force used is shown in Table 3.9. The variation in the string cutoffs arises due to the frequency response of the instrument body impulse response. A_{min} for each string is set to the minimum string cutoff from the 13 simulated plucks.

3.6 Final remarks on the constants

This chapter described the physical experiments I performed to measure the strings and instrument bodies. In total, 10 distinct instruments were measured: 5 violins, 2 violas, and 3 cellos. In order to facilitate gathering constants from multiple instruments, I chose to restrict myself to non-intrusive measurements (i.e. strings would not be removed from their instruments). Certain physical constants were not possible to measure without removing the strings, and thus were estimated from the other constants and recorded audio. In order to allow the simple homogeneous string algorithm to account for the measured string inharmonicity B , I permitted the estimated values for Young's elastic modulus E to be larger than would be realistic. Finally, I performed various simulations to pick constants which have no physical equivalent, such as the limited number of modes N or the sampling rate f_s .

The main research contributions of this chapter are:

- A method of estimating tension, linear density, and Young's elastic modulus of a string given its length, diameter, and recorded open-string plucks. The string tension was further calibrated using PID control to ensure that the simulated note had the desired pitch.
- An examination of the consequences of using different numbers of modes and sampling rates in the physical model, allowing researchers to choose the desired trade-off of audio quality and processing speed for these parameters.

3.6.1 More accurate measurements

As the focus of this dissertation is the virtual violinist, the measurements of physical constants was relatively crude. The resulting sound was acceptable, but the constants are likely not sufficiently accurate for any scientific simulations. In particular, the restriction on not removing strings from

their instruments meant that I could not measure the linear density ρ_L , which is an essential part of the frequency of a wave in a string ($f = \frac{1}{2L} \sqrt{\frac{T}{\rho_L}}$).

Measuring the string length L and diameter d could be performed with tools such as a digital calliper, but this is not a major source of uncertainty in the physical constants. Measuring other constants directly, rather than estimating them, is much more important. The linear density ρ_L is easily (albeit destructively) measured by cutting the string at the ends of the vibrating portion (i.e. at the bridge and nut) and weighing the result; alternatively, the entire string could be measured without any cutting, as the additional winding at the ends of the string will not alter the overall density by a great amount. Young's elastic modulus E can be tested with specialized equipment which applies forces to the material and compares the deformation.

As previously mentioned, string tension T falls somewhere between a physical constant (i.e. unchanging) and a musician-controlled variable. When strings are bought from a store, they are not under tension; musicians must choose the relevant tension themselves after placing the string on their instrument. This can be measured by clamping the string with a set of known weights on one end, and adjusting the weights until the desired frequency is reached.

Measuring string decays in the manner described in this chapter is slightly problematic. First, inducing a current via magnetic induction will dampen the string vibrations slightly; using an optical sensor would avoid this problem. Second, the string and electrical wires pick up EMF radiation from AC power and VLF transmissions. Third, plucking strings attached to the instrument body results in acoustical noise in the room being transferred to the string, in addition to the instrument body vibrating from the string vibrations and carrying those vibrations to and from the other three strings (sympathetic vibrations). Much more accurate measurements could be gathered by removing each string from the instrument body, clamping it heavily, and placing the apparatus inside an anechoic room to avoid stray acoustical noise.

This raises the question of the purpose of these measurements. For a scientific investigation of instrument acoustics, it is of course desirable to investigate each aspect separately (e.g., each string, the instrument body, the bridge, how energy is transferred between each element). However, that would necessitate major modifications to the physical modelling algorithm. Rather than treating the body as an LTI filter, the body would be a vibrating system; the string-body coupling would need to be modelled. Even the string algorithm would need changing; instead of treating the strings as homogeneous beams, the wound strings would be treated as a core plus one or more additional layers, each with its own set of vibrating modes.

Since the focus of this dissertation is the musical control of a virtual instrument, I opted to avoid that deeper layer of experimental physics and acoustics modelling. Instead, I deliberately measured the modal decays of the violin strings in place on the violin body in order to capture a small amount of the behaviour of a vibrating body via its effect on the string vibrations. This also made it practical to measure a wide range of instruments; with a bit of practice, I could "process" each instrument within half an hour. However, researchers interested in more accurate physical simulations should definitely perform more careful measurements. Borrowing instruments from friends and family is an inexpensive method of gathering constants, but this can add time pressure to return the instruments to their owners as soon as possible. Instead, I recommend entering into an agreement with a local instrument store, wherein researchers rent one instrument each day, thus giving ample time to perform the measurements and return the instrument to the store.

3.6.2 Final constants used

Table 3.10 lists the constants for each string which are used in the virtual instruments. As an aide memoire, the variables are: tension T , length L , diameter d , linear density ρ_L , Young's elastic modulus E , coefficients of static friction μ_s , dynamic friction μ_d , the slope of the hyperbolic curve μ_c , and the minimum vibration cutoff A_{\min} .

name	T (N)	L (m)	d (mm)	ρ_L (g/m)	E (GPa)	μ_s	μ_d	μ_c	$A_{\min} \cdot 10^3$
violin-e-I	73.8	0.319	0.33	0.41	62.50	0.89	0.26	0.20	0.25
violin-e-II	89.9	0.319	0.29	0.50	134.00	0.80	0.26	0.08	0.27
violin-e-III	104.4	0.321	0.28	0.57	186.00	0.66	0.27	0.09	0.19
violin-e-IV	75.1	0.319	0.32	0.41	75.10	0.71	0.31	0.12	0.22
violin-e-V	68.1	0.327	0.24	0.36	186.00	0.72	0.32	0.11	0.22
violin-a-I	59.2	0.325	0.60	0.72	19.50	0.90	0.25	0.19	0.98
violin-a-II	70.7	0.320	0.61	0.88	22.20	0.86	0.28	0.08	1.16
violin-a-III	46.7	0.320	0.75	0.58	4.92	0.65	0.28	0.09	1.06
violin-a-IV	67.9	0.318	0.63	0.86	23.10	0.69	0.33	0.12	1.19
violin-a-V	68.9	0.321	0.65	0.86	18.40	0.74	0.35	0.10	0.74
violin-d-I	57.5	0.322	0.88	1.61	4.56	0.91	0.24	0.20	1.10
violin-d-II	34.5	0.318	0.78	0.99	6.87	0.87	0.26	0.08	1.17
violin-d-III	51.0	0.323	0.74	1.42	12.90	0.69	0.26	0.09	0.73
violin-d-IV	34.6	0.321	0.62	0.97	18.30	0.70	0.34	0.12	1.10
violin-d-V	62.8	0.321	0.78	1.77	16.30	0.75	0.34	0.10	0.71
violin-g-I	45.1	0.325	0.85	2.79	4.79	0.88	0.26	0.20	15.40
violin-g-II	50.3	0.320	0.78	3.20	5.51	0.83	0.26	0.08	15.00
violin-g-III	59.3	0.326	0.65	3.65	19.00	0.68	0.27	0.09	15.90
violin-g-IV	56.9	0.322	0.98	3.58	2.84	0.74	0.33	0.12	16.30
violin-g-V	30.9	0.326	0.73	1.90	8.91	0.74	0.34	0.11	15.70
viola-a-I	87.0	0.367	0.36	0.82	81.30	0.81	0.23	0.25	0.40
viola-a-II	62.7	0.373	0.35	0.58	114.00	0.83	0.23	0.28	0.38
viola-d-I	57.2	0.363	0.44	1.26	55.30	0.74	0.22	0.25	5.40
viola-d-II	57.7	0.369	0.32	1.22	95.00	0.86	0.25	0.27	6.06
viola-g-I	43.2	0.361	0.67	2.17	8.01	0.79	0.23	0.25	5.71
viola-g-II	69.8	0.369	0.65	3.35	18.20	0.84	0.24	0.28	6.67
viola-c-I	45.0	0.369	0.72	4.87	18.10	0.77	0.22	0.24	9.20
viola-c-II	42.2	0.375	0.75	4.42	6.54	0.83	0.24	0.27	9.52
cello-a-I	139.6	0.658	0.75	1.66	25.00	1.16	0.36	0.17	2.01
cello-a-II	138.8	0.678	0.77	1.55	21.20	1.04	0.32	0.06	2.18
cello-a-III	138.1	0.687	0.77	1.51	22.50	0.96	0.44	0.09	2.26
cello-d-I	96.9	0.672	0.88	2.50	25.00	1.19	0.37	0.18	1.30
cello-d-II	99.1	0.677	1.02	2.51	21.30	1.10	0.33	0.06	1.35
cello-d-III	143.5	0.680	0.91	3.60	25.00	0.98	0.47	0.09	1.33
cello-g-I	102.7	0.662	1.21	6.16	8.60	1.20	0.37	0.18	1.38
cello-g-II	108.2	0.677	1.10	6.17	16.20	1.08	0.32	0.06	1.27
cello-g-III	142.2	0.688	1.08	7.85	22.20	0.95	0.44	0.10	1.32
cello-c-I	163.4	0.676	1.44	21.20	22.40	1.17	0.38	0.17	63.30
cello-c-II	134.9	0.678	1.54	17.50	13.40	1.05	0.34	0.06	63.80
cello-c-III	153.8	0.688	1.35	19.20	23.80	0.95	0.46	0.09	61.70

Table 3.10: Final constants used. Recall that the values of E are set artificially to ensure that the predicted string inharmonicity B matches the measured value of B .

Chapter 4

Implementation of Physical Modelling

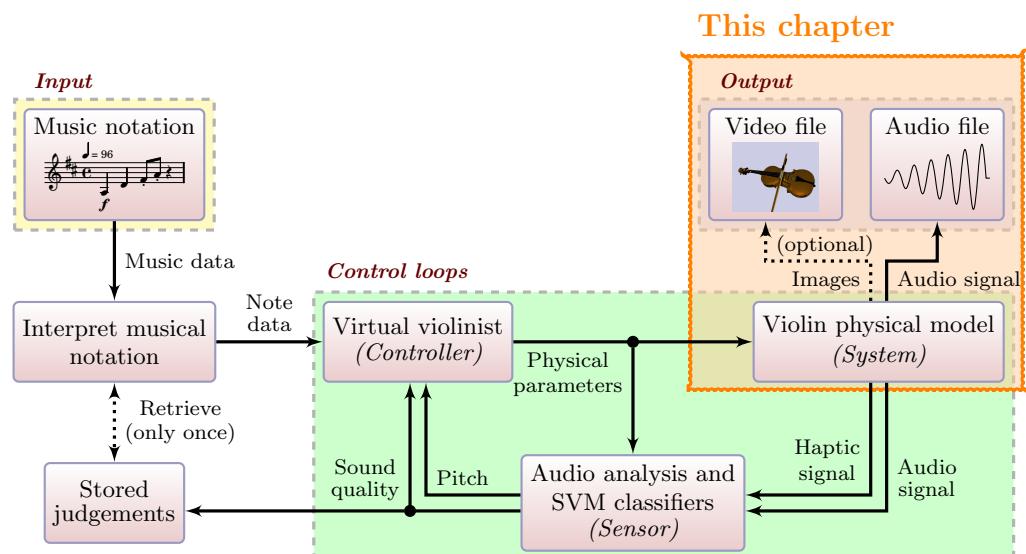


Figure 4.1: Context of this chapter.

In the past two chapters, I discussed the physical modelling equations and constants. This chapter discusses all remaining issues concerning the virtual instruments: Video output, issues arising in high-performance scientific software, and the Artifastring (“Artificial Fast String”) software project.

In order to visualize the interpretation performances of *Vivi*, the physical actions were animated to produce video output. The animations are created from the input physical actions, without using any of the simulations from the model.

The majority of physical modelling algorithms in acoustics research are written in high-level programming languages. Although this is appropriate for scientific investigations, it is not suitable for *Vivi*. Some aspects of my virtual musician training require thousands of seconds of simulated music and human input after every 10-30 seconds of simulated music. The time it takes to generate those simulations will greatly increase the burden of training *Vivi*. I therefore implemented the algorithm in a high-performance C++ library capable of generating audio 100 times faster than realtime.

```

### commands
#w      seconds
#b bow    seconds s      x_b      F_b      v_b      distance along bow hair
#f finger seconds s      1.0-x_f K_f
#p pluck   seconds s      x_p      y_p^d
w      0
### open strings arco
b      0.5    1      0      0      0      0
b      0.5    2      0.12   1.2    1      0.1
...
### scale pizz
f      3.5    1      0      1
p      3.5    1      0.25   1
w      3.5
...
### gliss pizz
f      18.18   1      0.1091  1
p      18.18   1      0.25    1
w      18.18
f      18.2    1      0.1097  1
w      18.2
f      18.22   1      0.1103  1
w      18.22
...

```

(a) Excerpt of `benchmark.actions`(b) Transcription of `benchmark.actions`

Figure 4.2: Benchmark test for physical actions. The actual `benchmark.actions` file is created by a python script and the exact durations of notes varies slightly from the sheet music transcription. The test is exactly 30 seconds long.

Audio 4.1: Audio generated from `benchmark.actions`

<http://percival-music.ca/dissertation/a.4.1.benchmark.wav>

There are two main ways that Artifastrings is used. The first is the `actions2wav` executable, which interprets an `.actions` text file and produces an audio `.wav` and haptic `.forces.wav` file. The second is as a C++ or python library, which provides an `ArtifastringsInstrument` object which may be manipulated.

Throughout this chapter, various tests and benchmarks are performed using the music shown in Figure 4.2, which also introduces the `.actions` format. This format is not intended to be written manually, but rather it is expected that other tools will output instructions in this format. The format is very easy to generate and parse; each action is a single line, # marks are comments, and the times of each action in seconds are in ascending order. Unless otherwise specified, all benchmarks in this chapter were performed on an Intel Core i5 running Ubuntu 12.04.1, compiling software with g++.

4.1 Video generation

In order to allow users to easily visualize the musical performance of *Vivi, the Virtual Violinist*, I added computer animation using Blender¹. Blender is an open-source computer graphics suite which provides python bindings for animation, allowing me to automatically convert any set of physical actions into computer animations.

4.1.1 Blender models

So far I have discussed a “model” in terms of physical modelling, i.e. a set of equations. In computer animation, the word “model” refers to a set of vertices and faces which defines one or more objects. These vertices can be manipulated with linear algebra to alter aspects of the scene (e.g., moving or rotating an object), and to render an image of the scene from any angle.

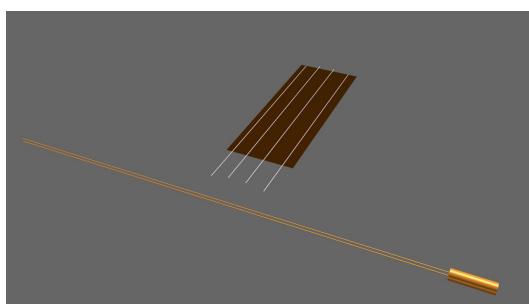
I created an extremely simple model of a violin, Figure 4.3a. The violin body (more accurately, the fingerboard) is represented by a thin rectangular prism; each string is represented by a cylinder; the bow is represented by three cylinders (a thick one for the frog, a thin one for the bow stick, and a thin one for the bow hair). Each finger is represented by an orange cone, while the plucking finger is a purple cone. In March 2011, I introduced my research to the lilypond-user@gnu.org mailing list, since *Vivi* uses LilyPond as discussed in Section 7.1. One reader of the mailing list, Marcos Press, was impressed with the sound synthesis and offered² to create a better model of a violin. I eagerly accepted his gracious offer, and he created Figure 4.3b.

Both models were integrated into *Vivi*. My animation system offers three levels of quality, which trade image quality against computation time. As an informal benchmark, I tested each quality level with the first 10 seconds of `benchmark.actions`. Level 0 uses the simple model and took 17 seconds, level 1 uses the full model and took 266 seconds, while level 2 uses the full model with additional tweaks (e.g., higher resolution, anti-aliasing, shadows) and took 3288 seconds. Level 2 uses HDTV resolution (1280x720), while levels 0 and 1 use 640x360.

¹<http://www.blender.org>

²<http://lists.gnu.org/archive/html/lilypond-user/2011-03/msg00670.html>

Subsequent private emails clarified that he placed his Blender model under the GNU Public License version 3.



(a) Simple model



(b) Full model by Marcos Press

Figure 4.3: Blender models of a violin. I created 4.3a, containing 532 vertices and 769 faces. Marcos Press created 4.3b, containing 132,265 vertices and 132,560 faces. Finger cones not shown.

Video 4.1: Three quality levels of animation

<http://percival-music.ca/dissertation/v.4.1.animation-quality-0.avi>

<http://percival-music.ca/dissertation/v.4.1.animation-quality-1.avi>

<http://percival-music.ca/dissertation/v.4.1.animation-quality-2.avi>

Communication between artist and programming

The models must contain specially named objects and vertices (or vertex groups). There must be a `violin` object, with 4 objects parented on the violin named `e-string`, `a-string`, `d-string`, `g-string`. Each string object must contain two vertex groups named `bridge-mark` and `nut-mark` to indicate the ends of the vibrating portion of the string. In addition, there must be a `bow` object parented on the violin. The bow must contain three vertex groups: `hair-frog` and `hair-point` to indicate the ends of the bow hair, and `wrapping` to indicate the bow stick. The orientation of the bow relative to the violin does not matter; the three named points on the bow are used to determine the orientation and the bow is rotated as necessary.

At the present time, I do not use different models to distinguish violin from viola or cello. Instead, I use different camera positions: The viola is slightly zoomed in (causing the model to appear bigger), while the cello is oriented in an upright position. In order to accommodate this, the models must contain at least three named cameras: `Camera.000` for violin, `Camera.001` for viola, and `Camera.002` for cello. Provided that these guidelines are followed, a 3D computer artist may create and use any model without needing to modify the animation scripts. The instrument may even be placed inside a larger scene in conjunction with other Blender animations.

4.1.2 Blender animation

As summarized in Section 2.4.2, there are three types of physical actions which are inputs to the model: moving the left-hand fingers, moving the bow, and plucking the string. The final “action” listed is to wait for m samples; for the animation, this simply specifies at which animation frame the action should occur. To simplify the rendering, when the bow is moving, the cone representing the plucking finger is removed; conversely, when a pluck is occurring, the bow is removed.

Finger actions

The physical model requires that the user specify the string and the relative distance from the bridge x_f . To find the desired placement of the cone representing the left-hand finger, the `bridge-mark` and `nut-mark` of the relevant string is retrieved. The apex of the cone is placed at the linear interpolation of those two vectors.

Pluck actions

A pluck begins in the same manner as finger actions; the user specifies the location x_p and the relevant point within the animation is found. However, after 100 ms have passed, the cone representing the pluck is gradually moved away from the string for the next 10 frames.

Bow actions

The bowing action places the bow on the string at position x_b , again finding the precise location by linearly interpolating between the two ends of the relevant string; this point is C_s . In addition, the contact point along the bow hair C_h is found by linearly interpolating between `hair-frog` and `hair-tip` according to the position along the bow hair given by the bow velocity. If no starting position along the bow hair is given, it is assumed that the simulation begins with the bow resting on

the string 10% away from the frog (roughly corresponding to the far edge of the winding around the bow stick). The starting position may also be specified directly.

The bow's orientation is found as follows. First, the intersection of `hair-tip`, `hair-frog`, and `winding` is found; this is point B_I . The normalized difference between `winding` and B_I gives the orientation of the bow hair to bow stick; this is \vec{B}_A . Second, I define \vec{B}_H as the difference between `hair-tip` and `hair-bow`. Third, the side of the bow which should be facing the left-hand fingers is found by taking the cross product of \vec{B}_A and \vec{B}_H .

The four strings are not oriented on a single plane; the two inner strings are farther away from the instrument body than the two outer strings. This displacement is necessary to allow the bow to play each string without touching the other strings. The bow hair must therefore be placed at a different angle for each string. For each inner strings, the angle is set to be parallel to the adjacent strings (e.g., when the bow is on the A string, the bow hair is parallel to a vector from the E to D string). These angles are A_D for the D string and A_A for the A string. I define $\Delta a = A_A - A_D$, then set $A_G = A_D - \Delta a$ and $A_E = A_A + \Delta a$. The actual bow angle of the currently-played string is A_b . Finally, the bow is translated and rotated to accommodate the point of contact along the string C_s , the point of contact along the bow hair C_h , and the bow angle A_b .

4.2 Implementation notes for physical modelling

There are a few issues to consider for efficient physical modelling algorithms: The effect of using discrete-time signals, the benefits of vectorized optimizations, and problems arising from subnormal numbers.

4.2.1 Discrete-time signals and music

To reduce the amount of overhead in function calls, the string simulation is generally calculated in buffers of M samples. For the specific case of the virtual string quartet, I usually process with buffers of $M = 512$ samples. This sets the usual control rate at $\frac{f_s}{512} \approx 43$ Hz, which gives durations of ≈ 23 ms. However, while a fixed control rate works well for interactive applications, I want to be able to simulate music whose note durations are not exact multiples of 512 samples. In such cases, I process the remaining samples with a partial buffer. Since the haptic output is downsampled by a factor of two, the number of samples must be a multiple of 2, and the amount of time represented by any function call to the physical model must be quantized to 11025 Hz, or ≈ 0.09 ms.



Figure 4.4: Music excerpt for buffer timing problems. When each note is quantized naively a sampling rate of 11025 Hz, the top whole note is 34815 samples, while the bottom sixteenth notes are 2175 each, or 34800 samples for one bar. The discrepancy of 15 samples results in the bottom part lagging behind the top part by 1.36 ms per bar.

Consider the music example in Figure 4.4; such a measure could easily be found in a Mozart piano sonata. Suppose there were 100 bars in that sonata; the two parts would become 136 ms out of phase. The Just Noticeable Difference (JND) for musical attacks is notoriously variable, but 136 ms is well beyond acceptable ranges for perceiving two events as occurring simultaneously (Wright 2008).

If this discrepancy is to be resolved, the software must keep track of the total simulated time as well as the sum of the durations of the desired notes. If the simulated elapsed time is less than the desired elapsed time, the next simulated note will be a few samples longer than it would otherwise be. The instrument body simulation code fills a buffer of memory provided by external code, rather than allocating user-oriented buffers itself. Communicating the size of the buffer back to user code is quite possible, but it adds an extra layer of complexity to both the physical modelling code and the user code. I therefore make no attempt to address the problem in the instrument body code. However, it is still appropriate to solve this problem for application programmers who do not wish to keep track of time themselves. I decided to implement the following: Any functions which contain a number of samples as an argument will create precisely that number of samples. The programmer is expected to keep track of elapsed time and make any desired adjustments to the number of samples. By contrast, if a user specifies a number of seconds using less direct method as discussed in Section 4.3.3, then the simulation software will adjust the number of samples.

4.2.2 Vectorized optimizations

In the past decade, CPU designers for desktop machines have stabilized the clock speed in the range of 1.5 GHz to 3 GHz. Faced with physical limits on the speed of signals from one portion of the silicon to another, recent efforts have focused on parallelism, both in terms of increasing the number of cores in a chip and Single Instruction, Multiple Data (SIMD). As the name implies, SIMD instructions load a number of variables into (a) large register(s), then apply the same calculation to all values.

Intel's Streaming SIMD Extensions (SSE) (Intel 2007) are the most widely used SIMD instruction set for desktop CPUs; AMD also supports SSE instructions. SSE provides eight 128-bit XMM registers, capable of performing calculations on 4 single-precision floats at once. Intel's new Advanced Vector Extensions (AVX), introduced in 2011, contains 256-bit registers, allowing 8 single-precision floats to be processed with one instruction. This does not mean that calculations will complete 4 times as quickly. Not all algorithms can be vectorized, and a vectorized algorithm will include elements which cannot be performed with SIMD instructions. However, a great deal of scientific and engineering computations can benefit from SIMD (Hassaballah et al. 2008).

One disadvantage of SIMD is that it can require writing code specifically tuned for the particular instruction set, both in terms of manually specifying the assembly instructions and also ensuring that data is aligned in memory in the expected manner. However, compilers have improved such that they can automatically detect some common instances of vectorizable code (Franchetti et al. 2005), and additional libraries have been written to simplify writing SIMD code. One such example is the Eigen library (Guennebaud et al. 2010), discussed in Section 4.3.1.

Informal testing with `benchmark.actions` showed that disabling explicit vectorization in Eigen resulted in the simulation running 2.4 times slower (0.96 seconds vs. 0.40 seconds). However, the FFTW library still used vectorized instructions as it was compiled with SSE support. With the 128-bit registers and using single-precision floats, the maximum speed-up would be a factor of 4; it is surprising that the vectorization helps as much as it does.

4.2.3 Subnormal numbers

The standard for floating-point computations in desktop computers is IEEE 754, first established in 1985 but revised in 2008 (IEEE Computer Society 2008). In order to maximize the benefit of SIMD instructions, I used 32-bit single-precision floats (referred to `binary32` in the 2008 standard). A value x is encoded in the form

$$x = \begin{cases} (-1)^s \cdot (1 + c) \cdot 2^{q-127} & \text{if } q > 0 \quad \text{"normal numbers"} \\ (-1)^s \cdot (0 + c) \cdot 2^{-126} & \text{if } q = 0 \quad \text{"subnormal numbers"} \end{cases} \quad (4.1)$$

where x is the final value, s is a 1-bit value representing the sign, c is a 23-bit fraction between 0 and 1, and q is an 8-bit unsigned integer. If $q = 0$ the exponent is set to be -126, and the floating-point numbers are said to be “subnormal” or “denormalized”. The range of subnormal numbers is $1.401 E-45$ to $1.175 E-38$. Due to the design of modern floating-point units, operations involving denormal numbers are significantly slower than normal calculations. The exact time penalty of denormal operations depends on the CPU, but it ranges from 1.4 to 520 times slower (Dooley & Kale 2006). On CPUs found in typical consumer use in 2012, the penalty is 5.5 to 92.2 times slower.

In most computations, values are never as small as 10^{-38} . However, these can easily appear in both DSP and physical modelling algorithms. In the DSP domain, non-stationary Infinite Impulse Response (IIR) filter with no input will gradually produce numbers which are smaller and smaller, eventually reaching denormals. In physical modelling, any damped vibrating structure — including vibrating strings — will decay into denormal numbers. For higher modes (i.e. $N > 35$), denormal numbers appear within 5 seconds of a pluck or lifting a bow off the string.

This performance penalty can be avoided by setting the processor mode to treat any denormal values as 0. In particular, setting the “flush to zero” (FTZ) and “denormals as zero” (DAZ) bits in the MXCSR control/status register of the SSE instruction set:

```
// C source code
_mm_setcsr( _mm_getcsr() | (1<<15) | (1<<6));
```

Informal testing with `benchmark.actions` showed that disabling denormal values results in the simulation running 6.6 times faster (2.64 seconds vs. 0.40 seconds). This is less than expected given the performance penalties shown in (Dooley & Kale 2006). There are two factors for this: First, even when a_n and \dot{a}_n for the higher modes are denormals, the lower modes will still have normal numbers which will pull the average time lower. Second, when the string vibrations are sufficiently low, the computations are disabled, as discussed in Section 2.1.4. This limits the amount of computations which would take place with small numbers.

4.3 Artifastring: Artificial Fast String

I created a software project to encompass all of my physical modelling work, called Artifastring (“artificial fast string”). Artifastring is freely available under the GPLv3, with all source code and data available as discussed in Appendix C. This project contains:

- SciPy/python scripts to analyze the data from physical experiments in Section 3.
- SciPy/python scripts to implement the physical modelling equations in Section 2, and to perform the simulations in Section 3.
- Blender models and python scripts to implement the video generation from Section 4.1.
- C++ library for a high-performance implementation of the virtual violin family.
- Stand-alone programs to perform the simulation and animation.
- A test suite of `.actions` files and documentation.

This section discussed issues arising in the C++ implementation, descriptions of the executable programs in Artifastring, performance benchmarks of the C++ code on common CPUs, and the use of Artifastring outwith of this dissertation.

4.3.1 C++ implementation

The C++ implementation was compiled with both `gcc` 4.6.3 (Stallman et al. 2012) and `clang++` 3.0 (Lattner 2005) to benefit from two sets of error and warning checks. It was also tested with the `valgrind` (Nethercote & Seward 2007) suite of debugging and profiling tools to ensure memory correctness.

In addition to the standard C++ libraries `libstdc++` and `libm`, this implementation requires `libfftw` (Frigo & Johnson 2005) and Eigen (Guennebaud et al. 2010). FFTW (the “Fastest Fourier Transform in the West”) provides highly optimized FFTs, used to compute the convolution with the body impulse response and haptic response. Eigen is a C++ header library, which requires the `.h` files to be present during compilation but does not require the library to be installed on the system during run-time.

Object-oriented programming

This implementation uses the object-oriented programming paradigm. There is an obvious metaphor between a real-world violin string and an `ArtifastringString`³ class. A violin string can have actions performed upon it — a left-hand finger can be placed upon it, it can be plucked, or bowed — and these actions are immediately added as class methods. Similarly, the internal state of the string — the modal values a_n and \dot{a}_n — are class variables. The instrument body receives a similarly simple translation from real-world physics to object-oriented design. The only class which has no obvious real-world equivalent is the FFT convolution. An UML class diagram is shown in Figure 4.5, while class variables of `ArtifastringString` are shown in Table 4.1.

³Notation: unfortunately in computer programming, the word “string” refers to a piece of text (“a string of characters”). To distinguish the virtual violin strings from the data structure representing text, I use the somewhat-clumsy word `ArtifastringString`

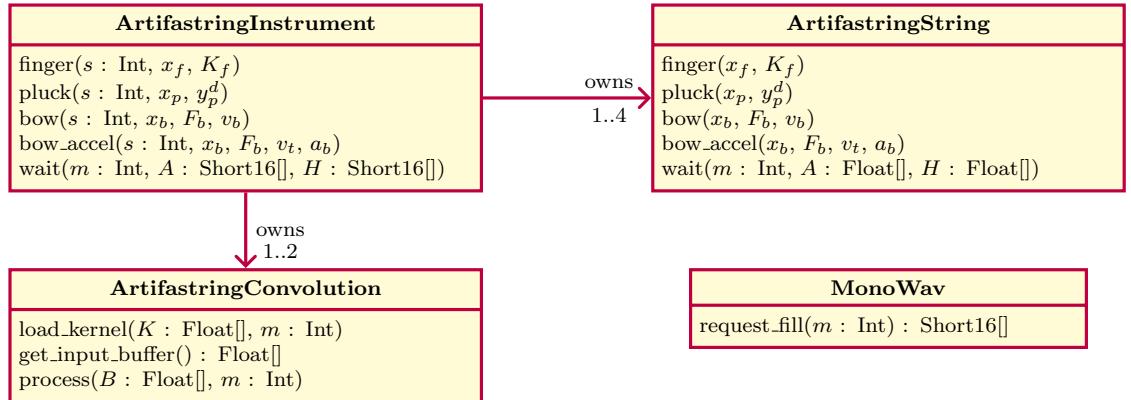


Figure 4.5: Class diagram for the Artifastring library (class variables are not shown). Unless otherwise specified, all variables are Floats. MonoWav is not directly connected to `ArtifastringInstrument`; it is merely made available in the library to allow client code to easily save data.

struct	explanation	variables in physical modelling
<code>sc</code>	string cached coefficients	$X_{1n}, X_{2n}, X_{3n}, Y_{1n}, Y_{2n}, Y_{3n}, G_n$
<code>va</code>	violinist actions	x_b (also used for x_p), $x_f, F_b, v_b, v_t, a_b, K_f$
<code>vc</code>	violinist cached coefficients	$x_0, x_1, x_2, \phi(x_0), \phi(x_1), \phi(x_2), D_1, D_2, D_3, D_4, D_5, D_6, D_7, D_8, D_9, D_{10}, D_{11}, K_0, K_1, K_2, R_0, R_1, R_2, y_p, y_p^d, t_p$
<code>ss</code>	string state	$a_n, \dot{a}_n, \text{slipstate}$

Table 4.1: Class variables of `ArtifastringString`. Variables are organized into structures. `sc` are cached values arising from the string’s physical constants and never change. `va` are the input physical actions, while `vc` are cached constants which only change a few times each second. Finally, `ss` contains variables which can change at every sample.

High-level C++ code with Eigen

One of the traditional disadvantages of C++ is that it is a low-level language. It grants the programmer a great deal of power with close interaction with the hardware at the expense of requiring extra care to avoid mistakes.

The Eigen template library for C++ (Guennebaud et al. 2010) mitigates some of the disadvantages of C++. Matrix arithmetic can be written easily without needing loops due to operator overloading. Template meta-programming in the library provides additional compile-time error checking and automatically determining the dimensions of variable-sized matrices. This ease of programming is nothing unusual in higher-level languages such as Matlab or SciPy/python, but Eigen brings this simplicity to high-performance C++. The Eigen library overloads operators `+` `*` when calculations involve vectors and matrices. In addition to making the code easier to read by avoiding loops, these operators include explicit vectorization. Part of the implementation of physical modelling is shown in Figure 4.6.

In addition, unlike some linear algebra packages, the additional “functions” which Eigen provides — overloaded operations, reductions such as `.sum()` — are not separate function calls, but are rather template expansions. This means that the memory cost of an Eigen array is simply the data stored (i.e. `float ah[40]` for a^h with 40 modes), with no run-time cost due to the arithmetic operations being expanded at compile-time. When Eigen vectors or matrices are created, the library ensures that data is stored in memory aligned: SSE instructions receive a significant performance penalty if the data is not 128-bit aligned.

```

inline float ArtifastringString::tick_pluck()
{
    // "hands-free" modes
    const AA ah = sc.X1 * ss.a + sc.X2 * ss.ad;
    const AA adh = sc.Y1 * ss.a + sc.Y2 * ss.ad;

    // "hands-free" string displacement under force locations
    const float y0h = (vc.phix0 * ah).sum();
    const float y1h = (vc.phix1 * ah).sum();
    const float y2h = (vc.phix2 * ah).sum();
    const float v0h = (vc.phix0 * adh).sum();
    const float v1h = (vc.phix1 * adh).sum();
    const float v2h = (vc.phix2 * adh).sum();

    // forces at those locations
    const Eigen::Vector3f matrix_b = (Eigen::Vector3f() <<
        -v0h * vc.R0 - (y0h - vc.y_pluck) * vc.K0,
        -v1h * vc.R1 - (y1h - vc.y_pluck) * vc.K1,
        -v2h * vc.R2 - (y2h - vc.y_pluck) * vc.K2
    ).finished();
    const Eigen::Vector3f Fs = inv_A * matrix_b;

    // apply forces
    const AA fn = vc.phix0 * Fs(0) + vc.phix1 * Fs(1) + vc.phix2 * Fs(2);
    ss.a = ah + sc.X3 * fn;
    ss.ad = adh + sc.Y3 * fn;
    return compute_bridge_force();
}

```

Figure 4.6: C++ code to process a pluck. **AA** is a typedef for a vector of size N (the number of modes), while the variables are explained in Table 4.1.

Parallelism and thread safety

Since the four **ArtifastringString** objects operate independently, there is an obvious potential for multithreading in order to take advantage of modern CPU design. Each string simulation could run as a separate thread, with a fifth thread acting as the instrument which combines the results of the four independent threads. Unfortunately, the synchronization cost caused such multithreading to be much slower than a single-threaded simulation. The **ArtifastringInstrument** only simulates a buffer of up to 512 samples at once, so more time was spent on mutex handling than on the actual string simulation. In addition, since strings are “turned off” when A_{\min} is reached, there are usually only one or two strings which require calculations to fill the buffer.

Fortunately, a great deal of the computation for training *Vivi* falls under the heading of “embarrassingly parallel problem”. Dozens of short simulations are performed, varying the parameters slightly, with no simulation depending on the results of another one during the same iteration. In this manner, *Vivi* can benefit from CPUs with multiple cores and hyperthreading, even though a single **ArtifastringInstrument** only uses one thread. However, in order to utilize multiple **ArtifastringInstrument** objects, the library must be thread-safe. This is not a problem for the Eigen library or my C++ code, but only small portions of **libfftw** are thread-safe. In particular, all memory allocation and FFT plan generation are not thread-safe. To resolve this, all portions of **ArtifastringConvolution** using such functions are protected with a mutex. Furthermore, plan generation produces some persistent data, which cannot be deallocated until all plans have been destroyed. A shared reference counter is used to determine how many instances of **ArtifastringConvolution** are in use; when this number falls to 0 the general FFTW cleanup function is called.

```

#include "artifastring/violin_instrument.h"
#include "artifastring/monowav.h"
int main ()
{
    // make objects
    MonoWav *wavfile = new MonoWav("output.wav");
    ViolinInstrument *violin = new ViolinInstrument();

    // get output buffer
    short *output = wavfile->request_fill(22050);

    // pluck the A string near the middle of the string with moderately strong force
    violin->pluck(2, 0.45, 0.8);

    // generate 1 second of audio
    violin->wait_samples(output, 22050);

    // automatically writes output file
    delete wavfile;
    delete violin;
}

```

(a) C++

```

#!/usr/bin/env python
import artifastring_instrument
import numpy
import pylab

# make objects
violin = artifastring_instrument.ArtifastringInstrument()

# get output buffer
output = numpy.zeros(22050, dtype=numpy.int16)

# pluck the A string near the middle of the string with moderately strong force
violin.pluck(2, 0.45, 0.8)

# generate 1 second of audio
violin.wait_samples(output)

# display time-domain output
pylab.plot(output)
pylab.show()

```

(b) Python. A direct translation of the C++ version (wherein the data is saved to a file) is possible, but it is more interesting to save the data to a `numpy` array for further processing or analysis.

Figure 4.7: Using Artifastring in C++ and python

Library usage and SWIG bindings for other languages

To support widespread usage, Simplified Wrapper and Interface Generator (SWIG) bindings (Beazley 2003) are also available. SWIG facilitates the use of C and C++ libraries in other programming languages such as python, lisp, R, and octave. Figure 4.7 shows typical usage in C++ and python. To facilitate easy integration of Artifastring with scientific python packages, I set the SWIG bindings to automatically replace any arguments of the form (`short *buffer, int num_samples`) or (`float *buffer, int num_samples`) with a `numpy` array.

CPU	Year	Clock	OS	time (seconds)			
				Violin		Cello	
				<code>g++</code>	<code>clang++</code>	<code>g++</code>	<code>clang++</code>
Intel(R) Core(TM) i5-2415M	2011	2.30 GHz	64-bit	0.40	0.44	0.28	0.31
Intel(R) Core(TM)2 Quad Q9550	2009	2.83 GHz	32-bit	0.66	0.66	0.48	0.48
Intel(R) Atom(TM) N570	2011	1.66 GHz	64-bit	2.60	3.39	1.73	2.36

Table 4.2: Benchmarks on common CPUs. The usual 30-second test `.actions` file was used. All computers ran Ubuntu 12.04.1. CPU frequency scaling was set to maximum performance. Each test was performed twice to ensure that the executable was cached, then the mean of the next three times was recorded. The compiler versions were the default for Ubuntu 12.04.1, namely `g++` 4.6.3 and `clang++` 3.0.

4.3.2 Benchmarks, profiling, and future improvements

Benchmarks were performed by running `actions2wav` on `benchmark.actions`. Artifastring was compiled with

```
 ${CXX} -fPIC -O3 -march=native -DNDEBUG
```

Results are shown in Table 4.2. A few of these results are worth examining in closer detail. Simulating a cello is considerably faster than simulating a violin due to the string sampling rates selected in Section 3.5.2: The violin uses M_f multiples of 4,4,2,2, whereas the cello uses M_f multiples of 2,2,2,1.

The effect of different compilers is interesting to note. `clang++` is part of the Low Level Virtual Machine (LLVM) compiler suite which arose from a 2004 research project at the University of Illinois. By contrast, the first release of `g++` was in 1987; the software architecture has grown “organically” over the past 25 years. This maturity can act as a two-edged sword: `g++` is very well-known and has received many optimizations over the years, but the design of `clang++` allows more experimental optimizations and provides much more informative error messages. The Atom processor is a low-power CPU designed for battery life in netbooks, so its relatively poor performance is expected. In addition, a significant amount of development for `clang++` and its related libraries comes from Apple. Apple does not use the Intel Atom processor in any products, opting to use ARM processors for their mobile devices and Intel Core CPUs in their desktops and laptops. It is reasonable to suppose that `clang++` has not received many optimizations for the Intel Atom processor. By contrast, the speed of `clang++` and `g++`-compiled code are identical for the Core 2. This processor has been used in many Apple computers, so it is plausible that `clang++` would have received more optimizations for this processor.

The benchmarks show that clock speed between different processors is decoupled from performance. In terms of SIMD instruction sets, the Atom and Core2 Quad support SSE 3, while the i5 supports SSE 4.2 and the newer AVX. However, the true benefit of AVX is not realized in Artifastring because Eigen does not yet support AVX.

Table 4.3 shows detailed profiling of time spent with `valgrind --tool=callgrind`. There are three aspects worth discussing: The overall efficiency of the implementation, directions of future optimization work, and implications for algorithm design.

To examine the efficiency of the implementation, I draw attention to two numbers in Table 4.3: The time to compute the FFTs involved in the body filters (30% in `libfftw`) and the time spent on low-level SIMD instructions (40% on `addps` and `mulps`). These present a rough estimate of the “fastest

Method or library	Time spent	Called	Rel. time ($\cdot 10^{-6}$)	addps	mulpss
<code>libfftwf</code>	30.77%	5684	5414	N/A	N/A
<code>AS::tick_release()</code>	27.38%	1654100	17	9.03%	12.74%
<code>AC::process()</code>	12.50%	2840	4401	0%	0%
<code>AS::tick_bow()</code>	10.45%	647540	16	3.11%	3.98%
<code>AS::tick_free()</code>	9.42%	1478428	6	2.31%	5.56%
<code>AS::tick_pluck()</code>	4.45%	206200	22	1.41%	2.24%
<code>AS::fill_buffer()</code>	1.90%	5680	335	0%	0%
<code>AI::handle_buffer()</code>	1.35%	1420	950	0%	0%
total	98.22%	-	-	15.96%	24.52%

Table 4.3: Profiling Artifastring to show the bottlenecks. Only the slowest 8 functions are shown. Benchmarking performed on the i5-2415M CPU with the additional compiler option `-g` to display detailed information about the time spent on each line of code. Abbreviations used for objects: `AS` is `ArtifastringString`, `AC` is `ArtifastringConvolution`, `AI` is `ArtifastringInstrument`. The `addps` and `mulpss` columns show the overall time spent on the two SSE assembly instructions (“Add Parallel Scalars” and “Multiply Parallel Scalars”), each of which operates on 4 single-precision floats.

The “main” function of `AS` is `AS::fill_buffer()`, which calls the appropriate `AS::tick_*`() function to calculate one time unit (dt) of output; this process is repeated as many times as is needed to fill the buffer (generally $512 \cdot M_f$ samples). The `tick_*` functions are as follows: `_bow` is called when the bow is touching the string, `_pluck` is called when there is a right-hand finger plucking the string, `_release` is called when there is only a left-hand finger touching the string, and `_free` is called when there are no external forces on the string.

The “main” function of `AI` is `AI::handle_buffer()`, which calls `AS::fill_buffer()` for each string, then sends the results to `AC::process()` (which in turn calls `libfftwf`).

possible implementation”: `libfftw` is renowned for its speed, and with one possible exception⁴ single assembly instructions are the fastest way to perform arithmetic on CPUs. This means that 30% of the processing time is set by the size of FFTs I chose in Section 2.1.5, and 40% of the processing time is set by the number of external forces (both finger and bow) chosen in Section 2.1 with the number of modes and sampling rates chosen in Section 3.5.1 and Section 3.5.2. Even if all other operations could be performed infinitely fast, it would only save 30% of the running time.

That is not to say that future improvements to the implementation are impossible. In particular, `AC::process()` currently does not use SIMD instructions for computing the overlap-add. Adding large arrays of floating-point values are a perfect use-case for SIMD instructions, so it is reasonable to expect that the 12.5% of time spent in this function could be reduced by a factor of 2 or more. Similarly, `AS::fill_buffer()` and `AI::handle_buffer()` might benefit from using SIMD for the $A[t]$ and $H[t]$ signals, in particular when combining signals of the same sample rate before passing the signal into the FIR filters, but these functions only take 1.9% and 1.35% of the processing time respectively. They are therefore not good targets for optimization.

The `AS::tick_*`() functions such as `AS::tick_pluck()` in Figure 4.6 cannot be optimized further without getting into very low-level SIMD design. It is appropriate to leave such optimizations to new developments in compilers and the Eigen library. The only worthwhile improvement would be to rewrite the calculation of “hands-free” string displacement and velocities to be a matrix multiplication rather than six distinct dot products. Matrix multiplication is a fundamental step in a large number of scientific and game/media computation, so implementing the physical modelling algorithm in such terms would allow me to benefit from future developments in compilers and CPU design.

⁴The exception comes in the form of the dot product command `dpps` added in SSE 4.1. Since a great deal of my `addps` and `mulpss` commands are used to perform dot products, a dedicated `dpps` may improve the speed.

Many applications of such physical modelling will only be concerned with the audio output $A[t]$ and not the haptic output $H[t]$. The calculation of $H[t]$ inside `ArtifastringString` comes “for free” as part of the normal simulations, but handling the $H[t]$ signals in `AI::handle_buffer()`, and most importantly the additional FIR filters in `libfftwf`, comprise roughly 20% of the computation time. If simulation time was a concern⁵, a command-line option (or even a compile-time `#ifdef`) could be added to disable the haptic output.

However, Artifastring is already very fast; on the laptop and desktop CPUs tested in Table 4.2, it ranges from 12 to 107 times realtime. Rather than increasing the speed of the implementation, it is likely that improvements to audio quality would be more useful. There are two easy ways of doing this: vary N based on the string rather than using a fixed $N = 40$ for all strings, and increase the sampling rate to $f_s = 44100$. Based on the values for B measured in Table 3.3, 40 modes results in a maximum modal frequency of 27 kHz for the violin E string but only 3.5 kHz for the cello C. Increasing N to 64 would give a C string with a maximum modal frequency of 7.4 kHz, while 80 modes would result in 10.9 kHz. However, the higher modes have much faster decays and thus have less effect on the simulation. Recall that due to string inharmonicity, doubling the number of modes will more than double the range of frequencies. If this is to be done, N would need to vary based on string: For the cello A string with 80 modes, the maximum modal frequency would be 20.8 kHz, which is well above the Nyquist frequency of 11.025 kHz. These concerns are even more relevant for the viola and violin.

This raises the question of sampling rate; maybe the instrument sampling rate should be 44100 Hz rather than 22050 Hz? In Section 3.5.2, only the cello C string has an f_s multiplier of 1; all other strings are simulated at $f_s = 44100$ Hz or $f_s = 88200$ Hz. The choice of 22050 Hz was partly motivated by a desire to reduce computation burden of analyzing the signals in the feedback control loop, but down-sampling could be performed during the analysis.

A few elements of the profiling suggest areas of the algorithm which could be improved. It was surprising to see that `tick_pluck()` and even `tick_release()` were slower than `tick_bow()`. This shows the effectiveness of Demoucron’s direct solution for the bow-string interaction, but also raises the question of whether a more elaborate bow-string interaction could be used. One option would be to improve the friction model such as using the double exponential friction proposed in (Smith & Woodhouse 2000). However, given the flaws due to “strongly rational” positions in bowing discussed in Section 2.3.1, a more important modification would be to extend the algorithm to take the bow width into account. Just as adding an extra finger position improved the simulation quality in Section 2.2.2 and Section 2.2.3, adding one or more bowing positions should mitigate the problems of “strongly rational” bowing positions. In addition, it would be desirable to simulate the finger with two positions during bowing as well as pizzicato, instead of switching from one finger force during bowing to two finger forces during string release and pizzicato.

⁵This is not likely to be a concern with desktop machines, but if the physical modelling were to be performed on a mobile device or embedded system, it is possible that removing haptic output could be the difference between realtime and non-realtime performance. Low-power CPUs designed for mobile phones or embedded systems may or may not include floating-point SIMD instructions, and are unlikely to have highly optimized FFT libraries. For example, the Raspberry Pi is a US\$ 35 general-purpose computer which uses an ARM11 CPU; this includes 32-bit integer SIMD, but no floating-point SIMD support. Floating-point SIMD was only introduced in the ARM NEON instruction set, which provides 128-bit registers just like SSE.

# bar	pitch	dur	# bar	pitch	dur
1.0	62	16	2.0	71	16
1.0625	62	16	2.0625	71	16
1.125	62	16	2.125	71	16
1.1875	62	16	2.1875	71	16
1.25	62	8	2.25	71	8
1.375	62	8	2.375	71	8
1.5	69	16	2.5	69	16
1.5625	69	16	2.5625	69	16
1.625	69	16	2.625	69	16
1.6875	69	16	2.6875	69	16
1.75	69	8	2.75	69	8
1.875	69	8	2.875	69	8

(a) Excerpt of `twinkle.notes`

(b) Excerpt of sheet music for "Twinkle, twinkle, little star"

Figure 4.8: Beginning of `twinkle.notes` used with `note2actions.py`. No feedback control was used to determine the bowing parameters; each note is hard-coded to begin with $F_b = 0.15$ N and decrease to $F_b = 0.1$ N, while the bow velocity accelerates at ± 30 m/s $^{-2}$ to $v_b^t = \pm 0.4$ m/s. These numbers were found empirically to produce a passable rendition of a young student playing “Twinkle, twinkle, little star”.

Audio 4.2: Audio generated from `twinkle.notes`

<http://percival-music.ca/dissertation/a.4.2.twinkle-no-feedback.wav>

4.3.3 Artifastring programs

Artifastring includes a few executable programs in addition to the C++ library.

actions2wav: This C++ programs reads an input `.actions` file and produces an audio `.wav` files and haptic output `.forces.wav` file. It is the main program in the suite. The user may specify the instrument type (violin, viola, cello) and instrument number. A simpler version which only produces audio output is written in python `play-file.py`.

actions2images.py, artifastring-movie.py: These python script reads an input `.actions` file and produces a series of images using the `blender` animation program. These are then combined with an audio `.wav` file to produce a movie with the `mencoder` movie encoder⁶. The overall quality, frames-per-second, and start/end frames can be specified. The latter option is used in *Vivi, the Virtual Violinist* to divide the video creation between multiple processes as a means of parallelizing the work. Recall that even the lowest-quality video generation is almost twice as slow as realtime when using a single thread.

notes2actions.py: This python script reads music in a MIDI-like `.notes` format and produces an `.actions` file for further processing. An example is shown in Figure 4.8.

research/: This directory and its sub-directories contains all the python scripts used in Chapter 2 and Chapter 3. One sub-directory of particular interest will be `numpy-string/`, an implementation of the physical modelling in `numpy` and `scipy`, allowing me to test and experiment with various aspects of model with the ease high-level Python code. Scripts devoted to analyzing

⁶`mencoder` is part of the `mplayer` movie player suite: <http://www.mplayerhq.hu/>

modal decays are in `mode-detect/`, analyzing impulse responses are in `impulses/`, and the least-squares fits to estimate unmeasured physical constants are in `make-constants/`.

`artifastring_interactive.py`, `artifastring_osc.py`: These python scripts allow interactive real-time control of the physical model, with the computer's keyboard or any networked input device via the Open Sound Control protocol. This functionality is discussed further in Section 4.3.4.

Use outwith this dissertation

Although this library was written for my research, it was designed and documented for general use for other projects. I hope that it can be of use to other researchers without detailed knowledge of acoustics or digital signal processing, so that more people can work on virtual bowed string performance. In the Spring of 2010, it was used as part of a second-year team design project between Electrical Engineering and Computer Science students at the University of Glasgow. The project created an artificial "hurdy gurdy" (a folk instrument from Eastern Europe) for use in contemporary music performances.

4.3.4 Realtime interactive use of Artifastring

The `artifastring_interactive.py` and `artifastring_osc.py` scripts present an `ncurses`⁷ text-based user interface which allows the user to modify violin performance parameters (e.g., string, finger position, bow force) and hear the resulting audio in realtime. The `artifastring_osc.py` script is also capable of receiving open Sound Control (OSC) messages (either locally or over a network), allowing the user to "perform" with the physical model using alternate input devices. Figure 4.9 shows `artifastring_osc.py` being controlled by an Android tablet.

⁷`ncurses` is a library for creating text cursor graphical interfaces, and was officially released in 1993 as a clone of `curses` library in the original BSD operating system.

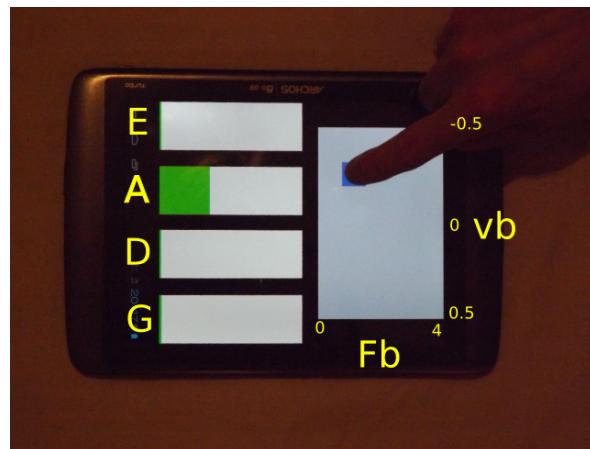


Figure 4.9: Interactive control of Artifastring with a tablet. The tablet sends OSC messages to a desktop computer or laptop which is running `artifastring_osc.py`. The four bars on the left control the string and finger position, while the large rectangle on the right controls the F_b and v_b bowing parameters. "Fb" and "vb" indicate the axes of bow force and bow velocity. The same interface controls all instruments; the text shows the behaviour for violin. When controlling a cello, the strings are ADGC, and the maximum F_b is 6 N. When controlling a viola, the strings are ADGC, while the bowing parameters are the same as for violin.

Video 4.2: Interactive control of Artifastring with a tablet

<http://percival-music.ca/dissertation/v.4.2.artifastring-interactive.avi>

There are three advantages to sending OSC messages from a tablet to a laptop for synthesis. First, having a “thin client” allows the interface to be ported and implemented on new systems much faster than a “fat client” which performs all computations. Second, laptops have a great deal more processing power than a tablet or cell phone. Third, the licensing requirements for some operating systems (notably Apple’s iOS) are incompatible with GPL software; splitting the functionality between the thin interface and the backend processing mitigates this legal burden. However, there is one large disadvantage of this method: It requires both a tablet and a laptop, making the system less portable than a laptop-only or tablet-only system.

In addition to being useful for demonstrations, the interactive scripts are useful for initial explorations of the parameter space. Although it is often more appropriate to use the `numpy` python implementation to evaluate bowing parameters and audio in controlled experiments, being able to quickly manipulate bowing and fingering parameters was an integral part of creating the virtual violinist discussed in Part II.

4.4 Final remarks on the virtual violin family

This chapter described Artifastring (“artificial fast string”), my implementation of the physical model in C++ along with the video generation using Blender. Artifastring is capable of producing audio and haptic output up to 100 times faster than realtime on common desktop computers.

The main research contributions of this chapter are:

- Applying the programming techniques of vectorized computing (SIMD) and avoiding subnormal numbers. These techniques are well-known to programmers, but it is useful to quantify their advantages for audio synthesis.
- Implementing the physical model in a fast, easy-to-use, open source library, suitable for either interactive use with humans or as a platform for virtual musicians. The efficiency of the implementation is shown with detailed benchmarks.

Unlike the previous two chapters, there were relatively few design choices in this chapter. The simulation could be represented using 64-bit floats instead of 32-bit floats, but I could hear no difference when evaluating this change; this is not surprising given that the final output is quantized to 16 bits and the bowing contains a stochastic term which likely overrides any randomness arising due to quantizing values to 32-bit floats. The simulation could be implemented in a different language, but the algorithm is simple enough that the ease of programming in a higher-level language gives no huge boost to programmer productivity, as the Eigen library already brings many advantages of high-level code to C++.

4.4.1 The model in general

Over the past three chapters I have described a physical model for violin, viola, and cello. I make no claim to complete mechanical accuracy; there are many known acoustic properties which are not included in this model as discussed in Section 2.4.1, and there are known inaccuracies in my measurements/estimations of physical constants which are examined in Section 3.6.1. However, for all its flaws, the physical model is far from useless: “All models are wrong but some are useful” (Box 1979,

p. 202). The intended use of this model is not scientific accuracy, but rather to produce audio which is sufficiently violin-like to serve as a platform for investigation of musical control.

Although the audio quality is sufficient for my needs, the musical abilities shown so far (`benchmark` from Figure 4.2, `twinkle` from Figure 4.8, and the interactive performance in Figure 4.9) has been rather poor. This is not any cause for concern: A violin has no musical ability. Musical performances arise due to the musician, not the instrument. The audio examples in this chapter were created without any control system, and thus do not indicate any flaw of the physical model. In fact, consider the most accurate physical model of all: A real violin. If we gave a violin to an untrained musician and asked them to perform some music, the resulting audio would sound much worse than the musical examples found in this chapter! This problem of musical control is precisely the problem my dissertation seeks to solve, so I consider these lack-luster musical examples as motivation for work on *Vivi, the Virtual Violinist*.

4.4.2 Public interface for `ArtifastringInstrument`

For the remainder of this dissertation, all interaction with the physical modelling takes place via the Application Programming Interface (API) for the `ArtifastringInstrument` C++ object. The interface is detailed in Table 4.4.

Function name	Type	Name	Input parameter notes
General setup			
<code>ArtifastringInstrument(...)</code>			
<code>InstrumentType</code>		<code>instrument_type</code>	The member of the violin family to simulate: <code>Violin</code> , <code>Viola</code> , <code>Cello</code>
<code>int</code>		<code>instrument_number</code>	The number within each instrument family. <code>Violin</code> : 0,1,2,3; <code>Viola</code> : 0,1; <code>Cello</code> : 0,1,2
<code>reset()</code>			
<code>get_physical_constants(...)</code>			
<code>int</code>		<code>which_string</code>	Instrument string number
<code>set_physical_constants(...)</code>			
<code>int</code>		<code>which_string</code>	Instrument string number
<code>String_Physical</code>		<code>pc</code>	String physical constants
Violinist actions			
<code>finger(...)</code>			
<code>int</code>		<code>which_string</code>	Instrument string number
<code>float</code>		<code>ratio_from_nut</code>	Measured as a fraction of string length.
<code>float</code>		<code>Kf</code>	How firmly the finger is pressed against the string. Normal finger strength is 1.0, while a light finger suitable for harmonics is 0.0001.
<code>pluck(...)</code>			
<code>int</code>		<code>which_string</code>	Instrument string number
<code>float</code>		<code>ratio_from_bridge</code>	Measured as a fraction of string length.
<code>float</code>		<code>pull_distance</code>	Measured in units of 5mm. This number was chosen so that a normal pluck on the violin uses a pull distance of 1.0. 0.0 produces no pluck at all.
<code>bow(...)</code>			
<code>int</code>		<code>which_string</code>	Instrument string number
<code>float</code>		<code>bow_ratio_from_bridge</code>	Measured as a fraction of string length
<code>float</code>		<code>bow_force</code>	Measured in Newtons
<code>float</code>		<code>bow_velocity</code>	Measured in meters / second
<code>bow_accel(...)</code>			
<code>int</code>		<code>which_string</code>	Instrument string number
<code>float</code>		<code>bow_ratio_from_bridge</code>	Measured as a fraction of string length
<code>float</code>		<code>bow_force</code>	Measured in Newtons
<code>float</code>		<code>bow_velocity_target</code>	Measured in meters / second
<code>float</code>		<code>bow_accel</code>	Measured in meters / second ²
Advance time			
<code>wait_samples(...)</code>			
<code>short*</code>		<code>audio_buffer</code>	Pre-allocated memory to hold requested number of audio samples
<code>short*</code>		<code>haptic_buffer</code>	Pre-allocated memory to hold requested number of haptic samples
<code>int</code>		<code>num_samples</code>	How many samples of 22050 Hz to advance. Must be a multiple of two, and the number of haptic samples is automatically halved.

Table 4.4: API for `ArtifastringInstrument`

Part II

Performing with the Virtual Violin Family

Chapter 5

Control loops

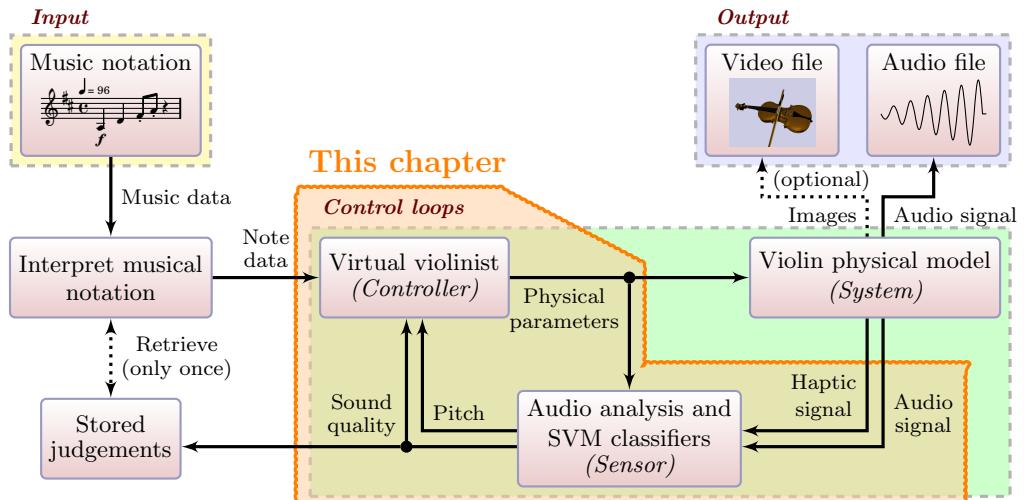


Figure 5.1: Intelligent feedback control in context.

In Part I of this dissertation, I described a physical model for instruments in the violin family. In Part II, I use that physical model to perform music with *Vivi, the Virtual Violinist*. The overall process is to use human input to calibrate various parameters of the virtual musician, then use those parameters to perform music.

While enabling *Vivi* to perform music, I attempted to keep the algorithm sufficiently general such that it could play any bowed instrument, be it a violin, viola, or cello. Each string has different physical characteristics; for example, as seen in Table 3.10, the violin-D-I string has a diameter of 0.88 mm and linear density of 1.61 g/m, while the violin-D-IV string has a diameter of 0.62 mm and linear density of 0.97 g/m. Those two strings will respond quite differently to the same external forces; if we desire to produce the same type of *forte* sound from both strings, then the bowing parameters must be adjusted accordingly. In order to accommodate all instruments without undue “hand-tweaking” of constants, *Vivi* must be able to automatically determine the appropriate values for each instrument.

This chapter begins by discussing the general approach to training *Vivi, the Virtual Violinist* with particular reference to human pedagogy, then discusses the two control loops: The right-hand bow control, and the left-hand pitch control. Chapter 6 will discuss calibrating initial parameters and interpretation of musical notation. Chapter 7 will discuss implementation details.

5.1 General approach to the virtual violinist

Violinist actions can be split into three categories: The left-hand fingering, the bowed *arco*, and the plucked *pizzicato*. The latter consists of discrete actions: Once the pluck-bridge distance x_p and string displacement y_p^d is chosen, the violinist has no further interaction with the instrument. By contrast, violinists often adjust the left-hand finger positions, and constantly adjust the bowing parameters. The majority of my attention therefore focuses on the bowing.

This section discusses four issues which guide the design of *Vivi*: The stochastic nature of the physical model; whether to draw upon detailed knowledge of the physical constants; scientific research about real-world violin control; and knowledge of pedagogical exercises.

5.1.1 Non-deterministic system

The bowing calculation from Section 2.1.3 includes a random term $u(t)$ which slightly alters the slope of the curve between static and dynamic friction in (2.36). This introduces an element of randomness which improves the simulation by adding noise at the slip-stick transitions (Chafe 1990, Demoucron 2008). However, this stochastic behaviour can cause large differences in the audio even with exactly the same bowing parameters being simulated. Figure 5.2 shows one such example where the model can randomly produce an acceptable or unacceptable bow-stroke. This randomness is mainly felt in the establishment (or not) of Helmholtz motion. Informal experiments showed that once a steady state is reached, $u(t)$ does not greatly alter the final result. For example, using the same parameters as Figure 5.2 but setting $F_b = 0.70$ N produced 10/10 notes with good timbre.

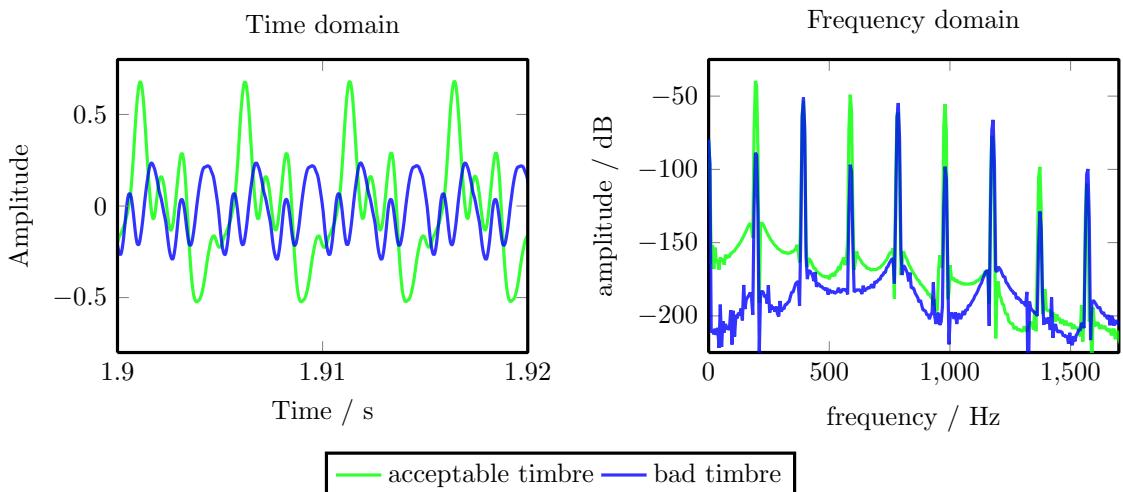


Figure 5.2: Different output with identical bowing parameters. Violin G string, bowed for 2 seconds with $x_b = 0.092$, $F_b = 0.25$ N, v_b beginning at rest (0.0 m/s) and accelerating to 0.4 m/s by 5.0 m/s². In the “acceptable” attempt, a good tone is established before the simulation ends; in the “bad” attempt, the string perceptually only vibrates as a “harmonic” (in the musical sense of the word). This perceptual judgement is supported by the frequency analysis, where energy is clearly focused in the even partials — the “bad” attempt has 50 dB less energy in the fundamental 196 Hz. Out of 10 simulations, an acceptable timbre was reached in 7 attempts.

Audio 5.1: Two bowing simulations with identical parameters

http://percival-music.ca/dissertation/a.5.1.accel-0_0_0.000_0.092_0.250_0.400_good.wav
http://percival-music.ca/dissertation/a.5.1.accel-0_0_0.000_0.092_0.250_0.400_bad.wav

A non-deterministic system adds an additional challenge to the control. This is not a problem with the model or the implementation; rather, this aspect of the model improves the realism of the challenge. Real-world instances of controlling a violin face even greater randomness: Mechanical actuators used for musical robotics do not deliver precisely the force or speed desired, while human violin students have far from perfect control over their muscles. With this in mind, any experiments to determine parameters for *Vivi* must be carried out multiple times to estimate the variance; parameters producing highly variable output should be penalized even if they occasionally produce good timbre.

5.1.2 Control theory and human control of violins

Having created and implemented a physical modelling algorithm, I will now treat the model (or “the system” in control terminology) mostly as a black box. Leaving aside *pizzicato* and finger position, there are three input signals (the bow-bridge distance x_b , the bow force F_b , and the bow velocity v_b) and two output signals (audio $A[t]$ and haptic $H[t]$). There are three reasons to discard any “privileged” knowledge of the physical modelling algorithm and physical constants.

First, as mentioned in Section 1.1.1, one of the potential applications of *Vivi* is to control musical robots. If my control system is designed explicitly for the equations in the physical modelling, then the control would not function properly when applied to real instruments. By only interacting with the system via the three input and two output signals, I hope to keep the control system more general, such that it may be applied to different physical modelling or even real-world instruments.

Second, human music teachers and students treat their instruments as black boxes. In the vast majority of cases, music students do not learn about Helmholtz motion, and certainly are not taught how to measure the linear density ρ_L of their instruments’ strings. That is not to say that music students are ignorant of how to manipulate their instrument — rather, they learn with metaphors and develop unconscious habits to improve their performance. I draw inspiration from these lessons in Section 5.1.3, as suggested by the goal of “human-like pedagogy” specified in Section 1.3.

Third, the system cannot be guided with control theory. This is not due to the particular equations used; the non-linear and stochastic bowing equation would be challenging but not impossible to handle. The insurmountable problem is the reference signal: We have no mathematical representation of a “good violin sound”. Simple classical control problems are concerned with a single value (e.g., room temperature, car speed). Complex modern control theory problems involve multiple values, but there is still a fixed goal to reach (e.g., maintaining a steady flying trajectory for an unmanned aerial vehicle, or the incredible descent of the NASA Curiosity rover without exceeding its allowable deceleration). By contrast, we know relatively little about which time-domain $A[t]$ signals could be considered to represent a “good violin sound”. We can specify very few guidelines on what $A[t]$ should contain. The signal should have a great deal of energy at the fundamental frequency f_0 of the desired sound (e.g., 440 Hz for the open violin A string)¹, and it should contain energy at near-multiples of f_0 (subject to string inharmonicity) — but should the first mode at 440 Hz contain more energy than the second mode at 880 Hz? In general, what should be the relative energies of the peaks? How much energy can be in subharmonics and non-harmonic peaks? As discussed in Section 1.2.5, objective analysis of violin sound is an active research area (Wrzeciono & Marasek 2010, Charles 2010), but it is a relatively new field. Furthermore, those efforts to characterize violin sound rely on machine learning rather than developing fixed rules which may describe $A[t]$.

¹Some instruments (e.g., timpani, bells) have a “missing fundamental” with little or no energy at f_0 .

Although I am not using knowledge of the physical modelling equations, the system will not be treated as a complete black box. I allow myself three pieces of knowledge about the system. First, the particular physical modelling I am using performs poorly at “strongly rational” x_b positions as discussed in Section 2.3.1. Deliberately avoiding those positions is an unfortunate deviation from the intended “black box” usage of the model, but this should not harm the generality of *Vivi*. Second, I also use the knowledge that my physical modelling algorithm performs much faster when x_b is constant, since modifying x_b requires recomputing $\phi_n(x_b)$. Third, with the exception of the above two points, I assume that the virtual violin behaves similar to real-world violins. Research on this topic was discussed in depth in Section 1.2.1, but I summarize the relevant facts here.

The main findings come from (Schoonderwaldt 2009): The bow-bridge distance and velocity set the overall amplitude, while the bow force sets the timbre. The three variables together determine whether Helmholtz motion can be achieved, but this is usually expressed in terms of limits on the bow force (Schelleng 1973). In particular, the allowable range of bow forces increases as the bow-bridge distance or velocity are increased. Detailed simulations (Guettler 2002) have shown that during the beginning of a note, the bow force and bow acceleration (instead of velocity) are the governing factors which affect whether Helmholtz motion is achieved. In musical terms, an extreme case of this would be an “on-the-string” attack contrasted with an “off-the-string” attack; the latter of which is much harder to perform with a good sound.

In more detail, the overall loudness (measured in dB) of the signal is very well correlated ($R^2 = 0.9$) with $\frac{v_b}{x_b}$. The spectral centroid of the sound increases with bow force and has no fixed relation with bow-bridge distance. This is contrary to “common wisdom” amongst violinists, who claim that moving the bow closer to the bridge will increase the “brightness” (spectral centroid) of the sound. The apparent increase in “brightness” arises from the violinist subconsciously adjusting the bow force in order to maintain Helmholtz motion while bowing close to the bridge.

One potentially surprising aspect of violin playing is that the bowing parameters can affect the pitch. (Schoonderwaldt 2009) notes that larger amplitudes of string vibrations results in a larger average tension in the string, causing an increase in pitch proportional to $\frac{v_b}{x_b}$. Another cause of higher pitches is string inharmonicity and bow force: As the bow force is increased, more energy resides in the higher modes of vibration, which are more affected by inharmonicity and thus give rise to a higher perceptual pitch. Finally, the physics of the stick-slip of the rounded corner of Helmholtz motion causes the pitch to fall; this is known in the literature as the “flattening effect” (McIntyre et al. 1983), and is seen with high bow forces, thick strings, and large bow-bridge distances. In total, these pitch effects have been observed to alter the pitch by -77 cents to +10 cents². (Demoucron 2008) showed that his model also produces pitch variation ranging from -30 cents to +20 cents for normal bowing, with the pitch dropping up to -70 cents when the maximum Schelleng bow force is exceeded.

²Cents are a logarithmic measurement system for musical pitch; given two pitches a and b , the cent $x = 1200 \cdot \log_2 \left(\frac{b}{a} \right)$.

5.1.3 Pedagogical inspiration

Since control theory cannot be applied to generate optimal bowing parameters, I turned to human pedagogy for inspiration. Beginning at age four, I learned cello with the Suzuki method (discussed in Section 1.2.6), and when I was in my early twenties I taught a dozen cello students. Some aspects of my experience as a student and teacher will inform the design of *Vivi*:

- Students do not practise alone; a parent should be involved in the daily practice.
- Students learn music from Suzuki violin book 1 and 2 (Suzuki 1978a, Suzuki 1978b), with a corresponding emphasis on Baroque music. Students are expected to spend approximately one year on each book. This provided me with a set of well-known music on which to train *Vivi*.
- The left-hand fingering is given in the sheet music; although beginning Suzuki students do not read sheet music, parents ensure that the student follows the printed fingering. Often coloured tape is placed on the fingerboard to indicate the proper placement of left-hand fingers. Many students do not adjust their finger placement if it is incorrect; this is a skill which is introduced slowly over the course of several years.
- The bow-bridge distances are set according to the dynamic. These are sometimes referred to as “bow lanes”, or the “Kreisler highway” (Collins 2009).
- The amount of bow for each note is often specified by the teacher (such as “half” or “quarter bow”), and the tempo is given by the piano accompaniment. Teachers sometimes direct students to imitate the teacher by playing music with the student (sometimes called the “mirror game”), in order to emphasize the amount of bow length to use.
- Students are taught to begin bowstrokes by pressing firmly into the string and moving the bow with a constant velocity.
- Students do not know physical values such as the friction coefficients (μ_d , μ_s) or the modal decay rates r_n of the strings.
- Students are not expected to give expressive music performances; a “robotic” performance³ is an acceptable place to start.

Left-hand finger x_g

For convenience, I define $x_g = 1 - x_f$. This allows us to discuss the finger position in terms of distance from the nut, measure the finger position as a distance away from the nut, rather than distance from the bridge, to match real-world pedagogy where finger positions are numbered from the nut. Since the sheet music specifies the finger position, we know x_g directly. To increase the “human-like” variability, I set *Vivi* to randomize the actual x_g with a normal curve. The standard deviation of this distribution can be altered to simulate the skill level of the student. Although beginning music students do not adjust their finger positions, *Vivi* will do so if the simulated skill level of the student is high enough. In this case, we can use a classical feedback loop as we have a reference signal (namely, the expected frequency of the note in the musical score).

³I do not encourage inexpressive music, but a virtual violinist can be useful even without expressive performances. Recall the example of Vocaloid from Section 1.1.2 and assistive technologies from Section 1.1.3: musicality can be added by the user while the software itself strictly follows the given instructions.

Bow-bridge distance x_b

Given the notion of “bow lanes” in pedagogy and the acoustics research of the effect of x_b , I will hard-code x_b based on the printed dynamic of the music. This imitates a student following the instructions of her music teacher with respect to bow lanes. In reality, maintaining a constant distance between bow and bridge is a difficult task even for students with 10 years of practice. However, for simplicity and speed of simulation I will allow *Vivi* to have such precise mastery over x_b . One set of pedagogical exercises for beginning violinists characterizes five different contact points (CPs) between bow and string. The sound arising from CP 1 (closest to the bridge) is characterized by “raucousness, grittiness”; CP 2 gives rise to a “resolute, vigorous” sound; CP 3 suggests “warm, sonorous”; CP 4 produces “dulcet, honeyed”; CP 5 gives rise to “ghostly, airy” sound (Collins 2009, p. 133).

Bow velocity v_b and acceleration a_b

If the amount of bow to use for a note and the tempo are set, then the average bow velocity is known. Average bow velocity is also the easiest bowing parameter to imitate — humans vision is quite good at spotting movement of 20–70 cm over a period of 0.2–2.0 seconds at a distance of 1–2 m. By contrast, viewing the bow-bridge distance (5–20 cm at a distance of 50–100 cm) is more challenging.

To imitate the teacher demonstrating the average bow velocity for the student, I will set the bow velocity v_b based on the dynamic of the music. In reality, a violinist will make tiny adjustments to v_b in order to maintain a good sound. However, for simplicity I will assume that such modifications do not occur. To increase the “human-like” variability of sound and mimic the untrained muscle manipulation of a beginning violinist, the actual bow velocity will vary slightly from the intended bow velocity. To mimic the “stop–go” bowing of students, the bow acceleration a_b will be relatively high. For simplicity, the acceleration will be constant until the target velocity is reached.

Bow force F_b

Other than the general direction of “press the bow firmly into the string before beginning a bow-stroke”, the general guidelines listed above do not include anything about bow force. Bow force is very difficult for students to observe; experienced violinists can estimate vague amounts of force by looking at the amount of curvature in the flexible wood of the bow stick, but such judgements are no more detailed than “light”, “medium”, or “hard”. Bow force is essentially taught as a free variable which violinists must manipulate in order to produce the desired timbre.

I take the same approach to *Vivi*. Since I am setting x_b and v_b based on the dynamic, it falls to F_b to vary in order to establish and maintain a good timbre. This raises the problem of judging the sound’s timbre: As previously discussed, there is no ad hoc objective rule for determining the quality of violin sound. I therefore turn to the first guideline of Suzuki lessons: “Students should not practice alone; a parent should be involved in the daily practice”. In reference to *Vivi*, I consider the computer to be the “student”, while the human user is the “parent”. Concretely, the user will judge the quality of simulated sounds. These judgements will be used to train a machine learning system, which will then be used to evaluate future simulated sounds and attempt to establish and maintain the desired timbre. After *Vivi* has produced more audio, the human user will listen to the resulting sound, and identify any problems. This cycle will repeat until the user is satisfied with the sound resulting from *Vivi*’s control of bow force.

5.2 Right hand: Sound quality and bow force F_b

As discussed in Section 5.1, we have no a priori mathematical description of violin sound quality. I therefore turn to machine learning as is usual in research on timbre, both violin-specific (Wrzeciono & Marasek 2010, Charles 2010) and for general audio analysis (Peeters 2004, Tzanetakis 2007). I cannot rely on machine learning trained on audio recorded from real violins, since there is no guarantee that the physical model will produce exactly the same output as a real violin given the same input.

Parameters x_b and v_b are specified from the dynamic as shown in Table 5.1. These values were determined by experimentation with the `artifastring-interactive.py` script; as stated in Section 5.1.3, the pedagogical inspiration is that the student simply follows the teacher’s instruction in setting these values. The remaining variable is the bow forces F_b . I again draw inspiration from human pedagogy for its control: A human “teacher” will give judgements about simulated audio from the physical model. Those judgements (or “classification labels” c) will be used to train the machine learning. There are 5 possible judgements about the sound, presented in Table 5.2.

The machine learning operates as follows. First, the human user provides judgements c about certain audio files. Second, the audio files (and related haptic signal files) are summarized with various metrics (“feature extraction”). Third, the machine learning is trained to recognize patterns linking the features to the judgements c ; when some new un-annotated data is presented to the system, that data is given a category based on the previously-trained pattern recognition. An example of the pattern recognition is given in Figure 5.4. I used one Support Vector Machine (SVM) per string per instrument type, for a total of 12 SVMs to process all violins, violas, and cellos. Each SVM is trained on 24 features (or “attributes”) with approximately 2000 instances.

Dynamic	Bow position x_b (fraction of L)	Bow velocity v_b (m/s)
<i>f</i>	0.092	0.40
<i>mf</i>	0.134	0.33
<i>mp</i>	0.154	0.26
<i>p</i>	0.186	0.20

Table 5.1: Physical parameters determined by dynamic.

Class c	Human judgement of sound	F_b in audio examples
1	not audible	0.010 N
2	“wisp” or “whistling”	0.052 N
3	acceptable	0.386 N
4	“harsh” or “detuned”	1.611 N
5	not recognizable as coming from a violin	5.560 N

Table 5.2: Human judgements of bowing. Informal experiments suggested that using 7 classes instead of 5 causes the human teacher to spend more time trying to decide between class x and y , without any noticeable improvement in the resulting controller audio output.

Audio 5.2: Examples of bow force classifications

```

http://percival-music.ca/dissertation/a.5.2.violin-0\_0\_0.000\_0.092\_0.010\_0.400\_1.wav
http://percival-music.ca/dissertation/a.5.2.violin-0\_0\_0.000\_0.092\_0.052\_0.400\_1.wav
http://percival-music.ca/dissertation/a.5.2.violin-0\_0\_0.000\_0.092\_0.386\_0.400\_1.wav
http://percival-music.ca/dissertation/a.5.2.violin-0\_0\_0.000\_0.092\_1.611\_0.400\_1.wav
http://percival-music.ca/dissertation/a.5.2.violin-0\_0\_0.000\_0.092\_5.560\_0.400\_1.wav

```



Figure 5.3: “Basic fingers” evaluated for training and calibration. Only one string for each instrument is shown; other strings used the same positions (i.e. 0, 1, and 6 semitones above the open string).

5.2.1 Human input: Classifying sound quality

The bulk of human input comes through an iterative process: *Vivi* plays some music (generally a scale or similar exercise), then the human teacher indicates problematic areas of poor sound. The machine learning is re-trained to take into account those problems, and the music is performed again, hopefully without making the same mistake.

Before this iterative process can begin, a certain amount of “basic” training must take place. The only firm requirements for this training are that there is at least one piece of audio labelled with each class c , that there are at least two different dynamics, and that there are at least two different fingers for each string. However, it is useful to include a few more examples in the “up-front” training. Notably, it is useful to focus attention on three different finger positions: Open string, low first position (the lowest non-open x_g , and extended third position (the second-highest x_g found in book 1 and 2 Suzuki). I avoided the highest position (fourth finger) due to the risk of confusion for human teachers between fourth position on one string and the open string above it. These notes are shown in Figure 5.3.

Unlike the iterative training where *Vivi* produces music to evaluate, the basic training occurs in an interactive script where the teacher can freely alter the bow force in order to produce the desired sound. Note that this “desired sound” does not only refer to a “good” sound (class 3, as per Table 5.2). In order to train the computer to recognize “horrible” sounds (such as class 5), the teacher must deliberately choose value(s) of F_b which produce that sound in order to correctly label it.

I define the “basic training” as at least 42 audio examples for each string.

- ***f*** and ***p***: All five classes for all “basic” finger positions. There must be at least 2 examples of “acceptable” sound, ideally denoting the highest and lowest F_b which produces an acceptable sound.
- ***mf*** and ***mp***: At least 1 example of “acceptable” sound for all “basic” finger positions, again indicating the highest and lowest F_b .

To reduce the amount of time spent training, I use one dataset per string of each instrument type, but share that dataset between instruments of the same type. For example, I use a single dataset to train *Vivi* for the D string of all 5 violins. The audio output of each violin will vary somewhat due to the instrument body responses, but the haptic output will be quite similar. In order to prepare the datasets, I require a few additional training examples:

- ***f*** and ***p***: At least 1 example of “acceptable” sound for each “basic” finger position.

These additional audio examples brings the total to 66 files for violin, 48 for viola, and 54 for cello. Each audio example is 9216 samples (≈ 0.42 seconds) long, and it takes approximately ten minutes to label all files for each string.

It should be emphasized that the audio in this “basic training” comes from a “constant” or “stabilized” state of the model. Here I use “stabilized” in a musical rather than control theory sense, in that the sound should have ceased to evolve. For category 3, the system will almost certainly be in stable oscillations (in the control theory sense); for category 5, the system should be “stable” in a state of noise. If the sound is evolving (i.e. the bow force has been increased and the timbre is changing) then the human teacher should wait a few seconds for the sound to settle.

Finally, it should be noted that the bow forces required to maintain a steady tone will not necessarily produce that same tone if the string is bowed from rest with those forces. For that reason, the bow forces of examples in this training set cannot be used directly to set the initial forces of bow-strokes.

5.2.2 Feature extraction and Support Vector Machine classification

The features are split into three categories: Time-domain analysis, spectral-domain analysis, and violinist actions. With two exceptions detailed below, the same features are calculated on both the audio and haptic signals. All features are implemented in the Marsyas library (Tzanetakis 2007). Feature extraction is performed on windows of 1024 samples with a hop size of 512 samples (≈ 23 ms), or an overlap of 50%. I require 1024 samples in order to generate reliable pitch estimates on the cello C string (lowest pitch 65 Hz, with the rule of thumb that it is desirable to include at least two periods of the lowest expected frequency).

The time-domain features are:

RMS: This feature is not normally used for audio and music machine learning, because of its sensitivity to the absolute amplitude of the signal, which can easily vary due to microphone placement. However, in my case, I am only interested in classifying artificial signals with no microphone involved in the processing chain, so this is a useful feature.

PeakToAverageRatio (PAR): This is also known as the Crest factor.

ZeroCrossing: Not used in haptic signal due to the DC offset rendering this meaningless.

The spectral-domain features are computed on only the first half of the spectrum (i.e. up to 5512 Hz for audio and 2756 Hz for haptic). This range contains most of the information, and avoids some features being influenced by the mostly-random information in the upper spectrum.

Centroid: The spectral centroid is the weighted mean of the spectrum, and is a good predictor of human perception of a sound’s “brightness”. It is used in the analysis of audio from real violins (Schoonderwaldt 2009).

Rolloff: The spectral rolloff is the slope of a linear regression line through the spectrum.

SCN: This is the normalized spectral centroid difference as defined in Section 2.3; it measures the distribution of energy between the first and second expected partials of the audio. This is a custom feature I added to Marsyas.

HarmonicStrength: The amplitude (in dB) of a few potential peaks in the spectrum. The amplitude is found by quadratic interpolation around the bin with the maximum energy within 1% of the expected frequency. In particular, I test $0.5f_0$, f_0 , and $2f_0$. This is a custom feature I added to Marsyas.

SpectralFlatness: The spectral flatness measure (SFM) indicates the “tonality” of a signal, as defined by the amount of energy which is concentrated in peaks. It is calculated by

dividing the geometric mean of the spectrum by its arithmetic mean.

RMS: Although the time-domain RMS is equal to the spectral-domain RMS by Parseval's theorem, recall that I am only considering the first half of the spectrum, so this limited-spectrum RMS differs from the time-domain RMS.

PeakToAverageRatio (PAR): When applied to the spectrum, the PAR gives more information about the concentration of energy.

The violinist actions are:

Finger: Left-hand finger position, in MIDI pitch values (semitones) above the open string.

Bow-bridge distance: Normal x_b value.

Bow velocity: Normal v_b value.

Although *Vivi* hard-codes x_b and v_b based on the dynamic while playing sheet music, these two parameters can be decoupled when using the control loop for other applications (e.g., interactive use with a tablet or other control device).

After extracting all features for each string, we have a set of 24-dimensional feature vectors (or instances) paired with their labels. For the specific case of the basic training of violin, each of the 66 files contains 9216 samples, for a total of 1056 instances from analyzing windows of 1024 samples.

More instances will be labelled during interactive training discussed in Section 6.1.

Machine learning

I used Support Vector Machine (SVM) classifiers. Intuitively, in two dimensions this training algorithm finds a line separating two regions. The line is chosen to maximize the margin between the line and the closest instances (known as the “support vectors”). The binary classifier SVM can be extended to cover multiple classes. With only two dimensions, as shown in Figure 5.4, a division line can be found “by eye”. However, the true power of SVMs is almost always realized by analyzing datasets with many features. The same mathematics which are used to calculate a maximum-margin separation line in two dimensions also applies to calculating a maximum-margin hyperplane in more dimensions.

Formally, an SVM is as a mapping between feature vectors $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, \dots, l$, with labels $y_i \in \{-1, 1\}$. This mapping is achieved by optimizing (Boser et al. 1992, Cortes & Vapnik 1995):

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^l \xi_i \\ \text{subject to} \quad & y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{5.1}$$

where $C > 0$ is the penalty for the error term and ϕ is a function which can map \mathbf{x} onto a higher-dimension space. \mathbf{x} is normalized such that the training data fits the range $[0, 1]$. The simplest SVM uses a linear kernel, $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$, but other kernels are also used, such as the radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$. I achieve good accuracy ($> 93\%$ in 10-fold cross-validation) with simple linear kernels, but I decided to use RBF kernels for additional accuracy ($> 98\%$) at the expense of one minute of additional computations whenever the SVMs are re-trained.

I make use of two extensions to the basic SVM algorithm which are implemented in `libSVM`. First, multiclass output between k classes (in my case, $k = 5$) is performed by training $\frac{k(k-1)}{2}$ binary

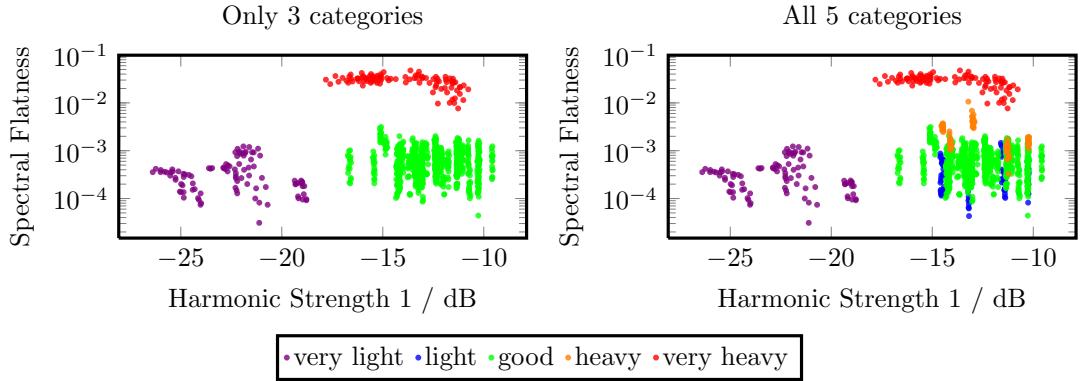


Figure 5.4: Bow force judgements with only two features, violin D string basic training. The two features are shown here before normalization. With only the 3 “extreme” judgements (left), it is easy to divide the two-dimensional plane into regions separated by lines with 100% accuracy. However, when all 5 judgements are considered (right), this two-dimensional plane is not sufficient to determine regions with acceptable accuracy. Training a SVM classifier on the right-hand data achieves 85.4% accuracy on 10-fold cross-validation, and omits any predictions of category 2 entirely. Differentiating between category 2 and 3 requires more than two features.

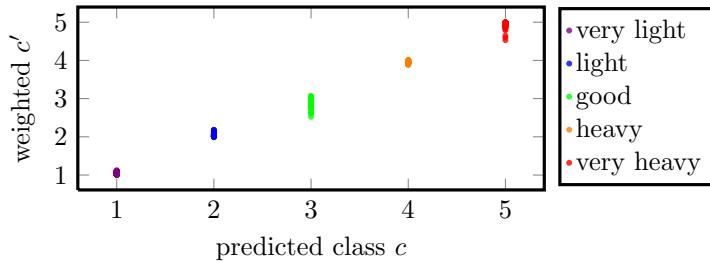
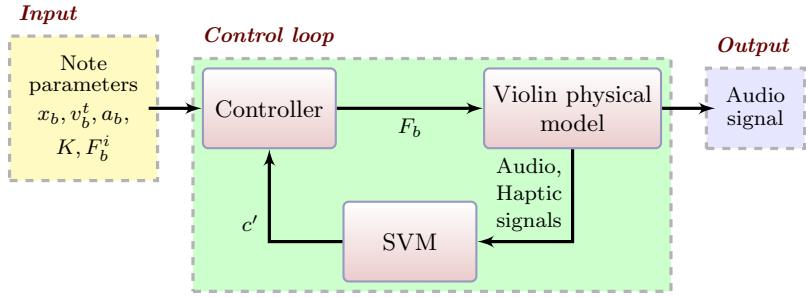


Figure 5.5: Comparison of direct SVM output and weighted probabilities, violin D string basic training. There is good agreement between the SVM output and the weighted c' . Since the initial labelling is not exact (it is not true that all $c = 3$ timbres are identical and could not benefit from any alterations to the bow forces) the slight spread of c' relative to c is not troubling.

classifiers, where the output of each binary classifier is treated as a vote in favour of that particular class and the class with the maximum number of votes is taken as the output of the entire classifier. Second, in addition to predicting a single class, libSVM also outputs probability estimates $p_i = P(y = i|\mathbf{x})$, $i = 1, \dots, k$. Additional details of both extensions are given in (Chang & Lin 2011).

During the training phase of the SVM, each class indicates a discrete value. However, when using the SVM to predict bow-force judgements, it is useful to receive continuous values to avoid an abrupt change from categories such as “needs more force” to “acceptable sound”. For this reason, I define c' as the weighted mean of class probabilities. Furthermore, I map the SVM judgements c' from $1 \dots 5$ to $-2 \dots 2$ to make it easy to recognize whether the bow force should be increased or decreased. This mapping is purely a convenience of notation in order to reduce the chance of errors when programming; it has no effect on the algorithm. Figure 5.5 shows good agreement between c' and c .

$$c' = 3 - \frac{1}{5} \sum_{i=1}^5 ip_i \quad (5.2)$$

Figure 5.6: Bow force F_b control loop.

5.2.3 Bow control loop

Once the SVM for each string is trained, *Vivi* is ready to play musical notes. A set of physical parameters is generated by the controller, and the physical model simulates 512 samples at 22050 Hz (i.e. a control rate of 43 Hz). Features are extracted from the 512 samples of $A[t]$ and 256 samples of $H[t]$ and are fed into the SVM classifier, which produces a judgement $c'(t)$. I refer to each unit of 512 samples as a *tick* of the control loop.

The bow-bridge distance x_b and target velocity v_b^t are set by the dynamic as shown in Table 5.1 and discussed further in Section 7.1, or by the user in the case of interactive control via a tablet as discussed in Section 7.3.4. To account for the varying weight of instrument bows, the bow acceleration a_b is set to 10 m/s² for violin, 7.5 m/s² for viola, and 5 m/s² for cello. Unless specified otherwise, each note begins and ends with the bow at rest ($v_b = 0$); the controller automatically decelerates the bow before the end of the note.

The bow force is adjusted during performance with the control loop shown in Figure 5.6. The standard technique for control is the Proportional-Integral-Derivative (PID) controller, where the input to the system is modified by some amount depending on the error between the desired output and the actual output. There are various techniques for tuning PID controllers such as the Ziegler-Nichols method (Ziegler & Nichols 1942). However, the weighted SVM output $c'(t)$ does not represent a firm error term as would be used in a classical control problem with a reference signal. I created my own method, taking inspiration from standard PID control. Examination of the effects of bow force (Schelleng 1973, Guettler 2002, Schoonderwaldt 2009), as well as casual experimentation of the physical modelling with the interactive script, suggest that timbre alterations occur with the log of bow force. I therefore modify F_b on the logarithmic scale. My control loop requires two parameters specific to the bow force which are both calculated in Section 6.1: The rate of change K and the initial bow force F_b^i . Other bowing parameters (x_b, v_b^t, a_b) are set directly based on the musical notation. The bow force is controlled by

$$F_b(0) = F_b^i$$

$$F_b(t+1) = \begin{cases} e^{\ln(F_b(t)) - K} & \text{if } t < W \\ e^{\ln(F_b(t)) - c'(t)K} & \text{else} \end{cases} \quad (5.3)$$

where W is an initial delay to allow the string to begin vibrating. In particular, $W = W_a + W_e$ where W_a is the number of ticks it takes for the bow to accelerate to the target velocity v_b^t and W_e is an extra delay.

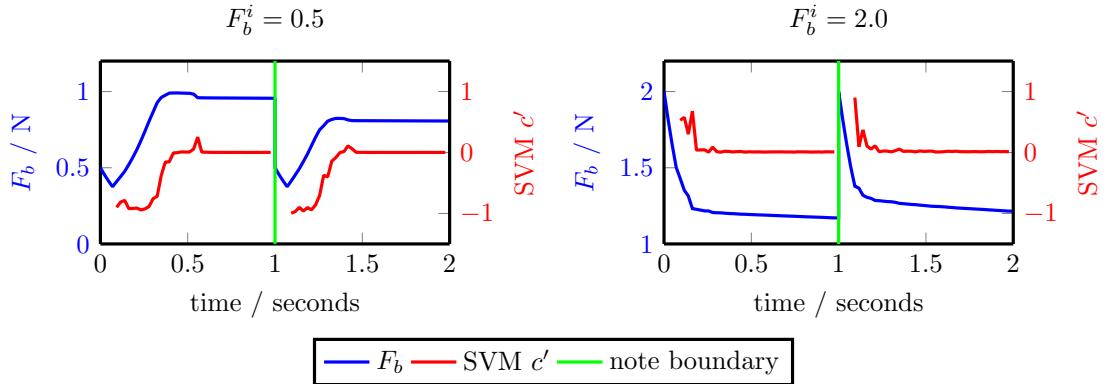


Figure 5.7: Two examples of bow control, violin f . Each example contains two notes, bowed at $v_b^t = \{0.4, -0.4\}$ m/s, with $x_b = 0.092, a_b = 10.0, K = 1.1$. The timbre judgements c' are withheld for the first W ticks of each note. The only difference between these two examples is F_b^i being 0.5 or 2.0 N. Note that the initial reduction of F_b is problematic when beginning with a low bow force; this is an accepted outcome of anticipating a relatively high F_b^i .

Audio 5.3: Two examples of bow control

<http://percival-music.ca/dissertation/a.5.3.force-correction-d-0.5.wav>
<http://percival-music.ca/dissertation/a.5.3.force-correction-d-2.0.wav>

I set W_e to 0 for violin, 2 for viola, and 4 for cello. These values reflect the time it takes for the lowest string of the instrument to respond to bow actions. Due to a combination of raw physical constants (chiefly the linear density ρ_L and friction coefficients μ_s, μ_d, μ_c) and the resulting frequency of vibrations (196 Hz for the lowest violin note, 65 Hz for the lowest cello note), the low cello strings are considerably slower to “speak”. It is not unusual for a low cello note to consist of 50-100 ms of shaped noise before the string “settles” into regular vibrations. This noise poses a problem for the machine learning, since the bow force judgements were given based on the *sustained* sound rather than the sound during the initial attack. For some portions of this initial “attack” sound, the SVM appears to fluctuate randomly between cases 2 and 4 (F_b being too low vs. F_b being too high). However, this is not a problem specific to the machine learning: Even a human musician with more than twenty years of musical training (me) was not able to reliably distinguish between all such cases during the “attack” portions. For this reason, I assume that during the note’s attack (i.e. $t < W$), F_b should be decreased slightly. This means that F_b^i will be slightly higher than would be necessary if we could categorize the required modifications to F_b during the initial attack, but this assumption is sufficient to realize student-level control of the bow. This assumption also plays nicely with real-world pedagogy discussed in Section 5.1.3. The assumed decrease in F_b lasts between $W = 2$ (43 ms) for violin p and $W = 8$ (186 ms) for cello f . These times refer strictly to (5.3); the time required to produce a musical “attack” (or for the note to “settle”) can be longer or shorter than W .

By default, F_b is only modified by (5.3) with no special provisions for the end of the note. However, when changing strings (i.e. suddenly removing the bow from one string) this can lead to a “pluck-like” sound. To avoid this problem, the note parameters can specify to lighten the bow force at the end of the note. The bow force begins to lighten at least 2 full ticks (46 ms) before the end of the note, and sets $F_b(t+1) = D_b F_b(t)$ with D_b being determined in Section 6.1.5.

5.3 Left hand: Pitch and finger position x_g

As mentioned in Section 5.1.2, the bowing parameters can alter the pitch. For that reason, if $|c'| > 1.0$, no adjustment is made to the left-hand finger position x_g since F_b is too volatile to make meaningful corrections to the pitch. Provided that F_b does not change a great deal, the pitch is primarily determined by the left-hand finger position x_g . Increasing x_g increases the pitch. There is a lag of a few milliseconds between the finger position and pitch, but we can essentially treat this as a purely linear system. This process is very similar to tuning the string tension described in Section 3.5.4.

The reference pitch P_r comes from the musical score. Depending on the desired musical interpretation, this pitch could be determined with equal temperament, Pythagorean tuning, or any other tuning system. This is discussed further in Section 6.2.1. Once P_r has been determined, it is converted into a MIDI pitch value from which is subtracted the base MIDI pitch of the relevant string M_s , producing M_r . This value is then converted into the initial finger position x_g .

$$\begin{aligned} M_r &= 69 + 12 \log_2 \left(\frac{P_r}{440} \right) \\ x_g &= 1 - \sqrt[12]{2}^{(M_s - M_r)} \end{aligned} \quad (5.4)$$

MIDI pitch values are used for their logarithmic nature, which is required during the conversion between a frequency in Hz and a finger position in relative string lengths. Although (5.4) makes use of an “equal temperament” interpretation of MIDI pitches, this does not presuppose the intonation temperament used to determine P_r . This x_g value is a credible estimate, but does not guarantee a note with perfect intonation. Although F_b does not vary enough to disrupt the feedback control of pitch, it does still present a bow-determined pitch offset between -30 to +20 cents which must be accounted for.

To accommodate this, I use classical feedback control, similar to the tension tuning performed in Section 3.5.4. A block diagram of the x_g control shown in Figure 5.8. Pitch detection was performed with the YINFFT algorithm, an extension (Brossier 2006) of the YIN algorithm (de Cheveigné & Kawahara 2002). The difference between the reference pitch and the output of YINFFT was taken modulo 12 to avoid octave errors; this value was further low-pass filtered with a median filter of length 3. This value becomes the error term $e(t)$, which is used in

$$x_g(t+1) = \begin{cases} x_g(t) + K_M e(t) & \text{if } |e(t)| \geq 0.01 \\ x_g(t) & \text{else} \end{cases} \quad (5.5)$$

$K_M = 0.01$ and the cutoff of 0.01 MIDI pitch units were set experimentally. An example of the

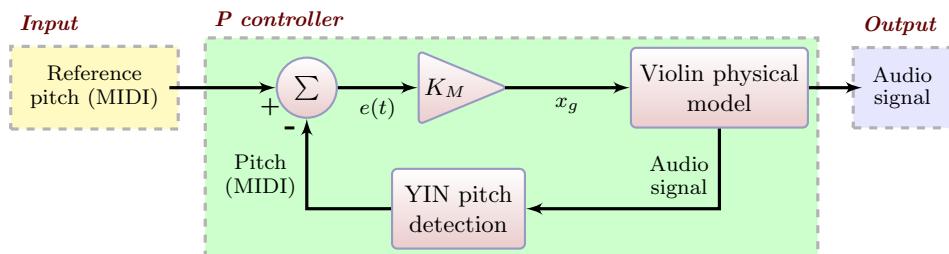


Figure 5.8: Tuning finger position control loop.

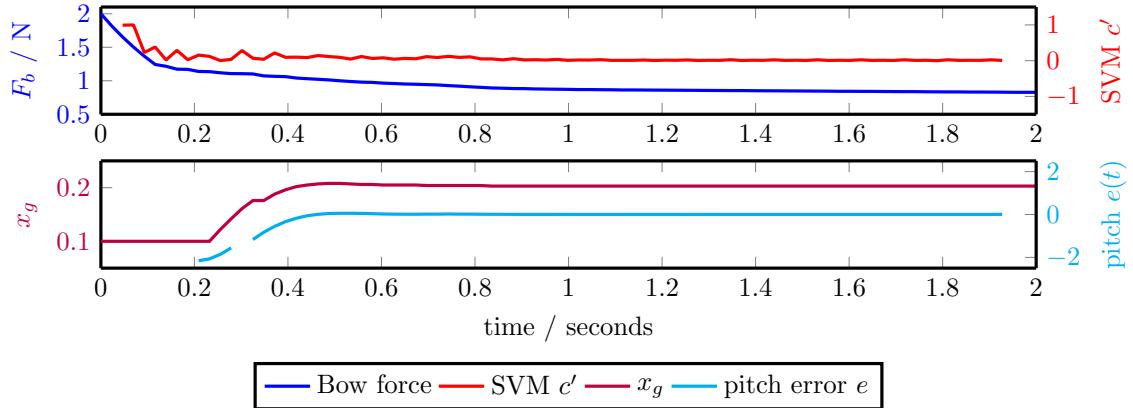


Figure 5.9: Examples of pitch control. The initial finger position x_g is deliberately much lower than the correct value to emphasize the adjustment. The pitch correction does not begin until the bow force judgements have stabilized and there are 4 ticks YINFFT output. The discontinuity at 0.3 seconds arises due to c' rising above 0.25.

Audio 5.4: Pitch correction

<http://percival-music.ca/dissertation/a.5.4.pitch-correction.wav>

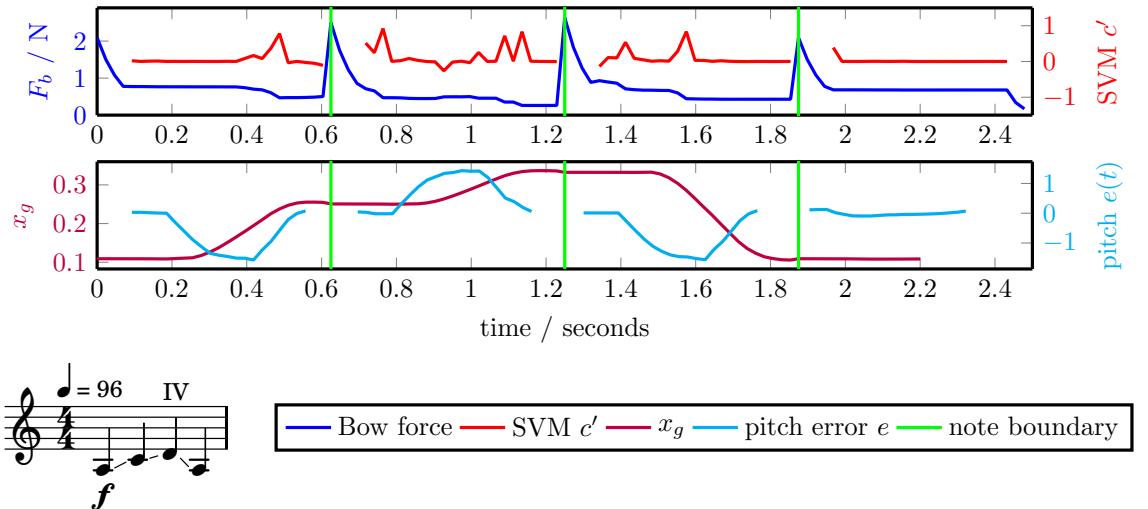


Figure 5.10: Examples of pitch glissando control. Bottom left: The musical score showing the notes and glissando.

Audio 5.5: Glissando control

<http://percival-music.ca/dissertation/a.5.5.violin-glissando.wav>

pitch corrections is shown in Figure 5.9.

The same feedback loop can be used to perform *glissando* notes (a gradual change of pitch between two notes). An example is shown in Figure 5.10. To play a *glissando* between two notes, the beginning and ending reference pitches are given. The reference pitch P_r used throughout the note is linearly interpolated between those values.

Musicians will alter the beginning and ending times of a *glissando*: Sometimes they will begin to shift the left-hand finger as soon as the glissando note begins; other times they will wait short period before beginning to move. Similar concerns apply to the ending of the finger movement. I arbitrarily set $Vivi$ to begin moving the finger after 8 ticks, and to reach the final pitch 8 ticks before the ending of the note, but these figures could be increased or decreased if a different playing style was desired.

5.4 Final remarks on the control loops

This chapter described the general approach to the physical actions performed by *Vivi, the Virtual Violinist*, the intelligent feedback control used to establish and maintain the tone quality, and the classical feedback control for the pitch.

The main research contributions of this chapter are:

- Applying knowledge of human control of violins and violin mechanics to the physical model of a violin, bridging the gap between music, acoustics, and control theory.
- Creating a method for non-technical musicians to train the computer to recognize good timbre, then using the trained system to control the virtual instrument.
- Generating a mid-level note representation which supports all notation found in easy violin music (Suzuki books 1–2). This allows the score to be decomposed into individual notes with no loss of musical information, and allows the low-level control loops to operate on discrete notes.

Many design choices were made in this chapter; most of these were fairly arbitrary and could benefit from an in-depth examination of the alternatives. In particular, the choice of features to use and the type of machine learning was motivated by the features and SVM already being implemented in Marsyas; the system appeared to work correctly so I did not investigate other machine learning algorithms. If the accuracy of the machine learning was not as high as it was (above 98% with 10-fold cross-validation), I would have investigated using other features and/or algorithms, such as those used in (Charles 2010, Chudy & Dixon 2012).

The choice of many parameters was informed by a mixture of acoustics research and casual listening. For example, the bow-bridge distances and velocities were guided by the knowledge that loudness is correlated with $\frac{v_b}{x_b}$, but the precise values were chosen through informal experiments with the `artifastring_interactive.py` script. Other researchers may wish to choose different values, or better yet, conduct formal listening experiments to evaluate the entire range of parameters and audience perception.

The remainder of this section discusses plucked notes, a few limitations of the machine learning and bow control, prospects for future improvements of the bow control, and finally outlines the API which allows the higher-level parts of *Vivi* (namely, the interpretation of musical notation) to interact with the low-level control loop.

5.4.1 Plucked notes

Plucked notes (known as *pizzicato*) in *Vivi* make no use of either control loop. The string excitation involves a single impulse, so there are no parameters which can be adjusted to alter the sound quality. Real musicians will occasionally alter the left-hand position to fix intonation mistakes in plucks, or alter the strength of the right-hand plucking action, but these are relatively advanced techniques which are left for future research.

In terms of the low-level C++ interface, the pluck position is arbitrarily set to $x_p = 0.37$. The pull distance is set to $0.2F_b^i$ for plucks of an open string, and $0.3F_b^i$ for plucks of a fingered string ($x_g > 0$). Open-string plucks are less damped than fingered-string plucks, so reducing the pull distance in this manner produces notes of perceptually equal loudness.

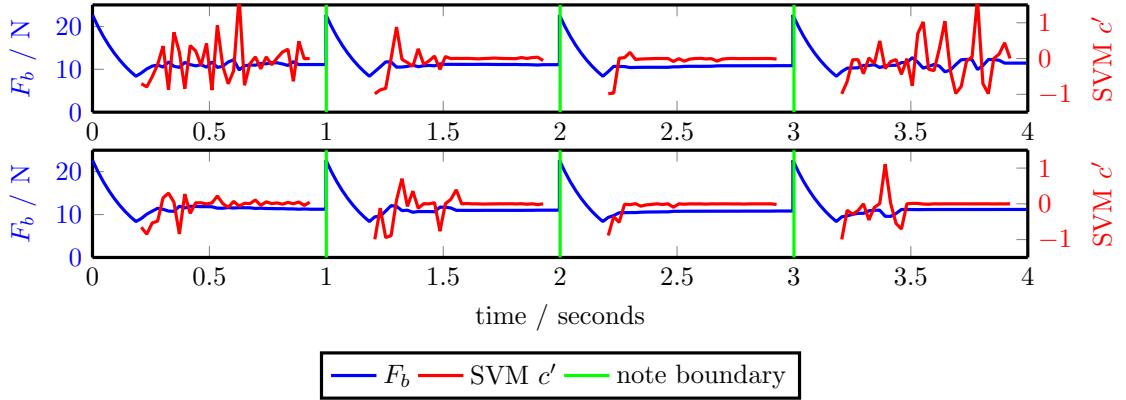


Figure 5.11: Problematic control of the cello C string. There are two examples, each containing four notes with the same parameters: $x_b = 0.092$, $F_b^i = 22.658$ N, $K=1.13$, $v_b^t = \pm 0.4$ m/s (alternating downbow and upbow). The internal string state is not reset after each note. In the first example, the first and last notes do not sound good, while the inner two notes sound good. In the second example, the first note is acceptable while the remaining three notes sound good.

Audio 5.6: Problematic control of the cello C string

http://percival-music.ca/dissertation/a.5.6.cello-0-1.13_0_0.000_0.092_22.658_0.400_1.wav
http://percival-music.ca/dissertation/a.5.6.cello-0-1.13_0_0.000_0.092_22.658_0.400_2.wav

5.4.2 Limitations of the bow control

As was noted in Section 5.1.1, the physical model includes a stochastic element which can cause the difference between reaching a steady state of acceptable sound or not. One of the goals of the bowing feedback control is to bring this stochastic element under control: F_b should be modified to cause the string to produce the desired sound. The bow control works well for the violin, but has difficulties with the cello C string. Out of the eight attempts to bow the open C string in Figure 5.11, only six examples produced a good sound, while one more example produced an acceptable sound. The difficulties with the C string are unfortunate but not unexpected: In the Suzuki repertoire, cello students do not play the C string until book 3 (where each book is expected to take one year of study). The cello C string is the thickest and heaviest string, and is difficult for humans to play as well as virtual musicians.

There are three avenues of future research which could improve the performance of the cello C string: Modifying the physical model to include an extra dimension for the string vibrations, changing the machine learning, and extending the bow control to include bow velocity.

The physical model only allows transverse vibrations (and thus displacement) in the string. This is at odds with the actions of real cellists: When playing f on the cello C string, a cellist presses the bow very firmly into the string (displacing it 1–2mm downwards of its rest position), then begins to draw the bow sideways (displacing the string 1mm before the bow begins to slip). Once so displaced, the string has a great deal of restorative force when it begins to slip, and the resulting motion is closer to the intended Helmholtz motion.

The bow control might benefit from more advanced digital signal processing, such as human perceptual processing (e.g., mid-ear filtering, mel frequency scale) and harmonic processing (e.g., harmonic spectral deviation, total harmonic distortion). In addition, the machine learning could be improved with ordinal regression (Li & Lin 2007), wherein the discrete classes (5 in my case) are not treated as independent labels, but rather as a ranking (i.e. label 4 is higher than label 2). This adds additional information about the problem which can improve the predictions. Although the machine

learning has good accuracy (98%), there are two risks which are not accounted for in that figure: First, the training/test data may not adequately represent the range of signals produced by the model while performing music, and second, this accuracy could represent overfitting. Although I am confident that the steady-state behaviour is well-represented by the human-labelled examples, I will not make that claim about the transient behaviour during the note attacks. One of the chief difficulties concerning note attacks is that the human judgements could vary. In particular, I gathered a few examples of “scratchy sound” produced from the string not quite reaching steady Helmholtz motion. One such example is the first note in Audio 5.6. I tried to label such examples once a day for four days to specify whether F_b should be higher or lower, but found that my judgements about that example were not consistent⁴.

In fact, if I as a cellist heard Audio 5.6 from my C string, my first reaction would probably⁵ be to alter the bow speed. Due to the elasticity and vibrations of the bow hair, musicians have more direct control of the bow speed rather than the bow force. Although the discussion of pedagogy in Section 5.1.3 noted that the *average* bow speed is specified by the teacher, a live musician could still alter the *local* bow speed during the note attack, then adjust the bow speed during the sustained portion of the note to compensate. Controlling both bow velocity and force would be a good step, but this poses a problem for human labelling. Judging audio on a two-dimensional scale would be too complicated. It would be easier for the human teacher to rank audio as “good”, “bad”, or “very bad”. However, the next step of the process is not obvious. If the machine learning correctly judges a particular tick of audio to be “bad”, should the controller increase or reduce bow force, increase or reduce bow velocity, or both?

One solution would be to give up on the principle of “human-like pedagogy” specified in Section 1.3 and use privileged information about the model. Once a particular piece of audio was ranked as “bad”, the computer could save the internal string state a_n and \dot{a}_n . A brute force approach could then decide what modifications were appropriate. The computer could then try all possible modifications to the bowing parameters from that string state, simulate for 20ms, then evaluate whether the audio quality improved. If there are 9 possible modifications (bow velocity higher, neutral, or lower; bow force higher, neutral, or lower), this additional layer of training would not take too long (although it may be desirable to repeat these simulations additional times to account for the randomness). Once the appropriate behaviour was determined, the computer would label that audio, then train a second layer of machine learning to recognize the audio as one of those 9 categories. This could be thought of as approaching the “subconscious” nature of real musicians bowing: The human teacher specifies whether the sound is good or bad, then the underlying layer figures out what bowing modifications are needed to reach that sound.

However, it would be unfortunate if the virtual musician used the trick of setting the string state directly. To retain the generality of applying to musical robotics, the computer could randomly choose one possibility, instead of simulating all possibilities by artificially setting the internal string state. This would require hundreds or even thousands of repetitions of notes in order to establish a sufficiently large database to make acceptable judgements about the best way to alter the bowing parameters.

⁴I hold a Bachelor of Music in cello and viola; if I cannot give consistent judgements for a specific set of examples then I cannot expect the computer to reach better answers through machine learning. Machine learning depends on good input; if the initial labelled data is flawed, then we should not be surprised if the output is imperfect.

⁵Bow control is largely subconscious, so I cannot state for certainty what I would do. A proper investigation would require additional hardware to monitor bowing actions, such as one of the systems developed in (Young 2007, Rasamimanana 2008, Demoucron 2008, Schoonderwaldt 2009, Maestre 2009, Pérez 2009).

The number of simulations required in either case is large, but there are two factors to bear in mind: First, this is a massively parallel problem with minimal information exchange required, so it could easily be run on multiple CPUs or even multiple computers. A single computer could generate over three thousands of hours of simulated music overnight. Second, real-life musicians develop their subconscious bowing ability over the course of thousands of hours of playing their instrument; if seems appropriate if high-quality control of a virtual violin requires the same amount of “practicing”.

5.4.3 Extending the control loops

There are a number of violin techniques which are not included in the control loops. These techniques were omitted because they are not taught to beginning students, and are not part of Suzuki books 1–2. I will give an overview of how they could be implemented.

Double stops: The physical model supports bowing two strings at once. The difficulty is in adjusting the bow force. There are two methods for this. The first method is to perform sound-source separation: Split the audio and haptic output into signals coming from each string. This is a non-trivial problem in DSP and machine learning, but given the limited range of signals which would arise from each string it could likely be performed with acceptable accuracy. The second method is to “cheat” by altering the physical model: Instead of generating combined signals $A[t]$ and $H[t]$, the model could output one pair of signals for each string $A_0[t]$, $H_0[t]$, $A_1[t]$, $H_1[t]$, $A_2[t]$, $H_2[t]$, $A_3[t]$, and $H_3[t]$. Once separate signals for each string have been gathered (however it is done), the bow control loop can operate as usual. Rolled chords can already be performed with the control loops; simply split the chord into distinct (short) notes, in order from lowest to highest string (or vice versa), holding the final note for most of the duration of the chord.

Pizzicato double stops appear in Suzuki book 2, but bowed double stops are not in the repertoire until the end of book 3. Bowed rolled chords are introduced in the same piece of music.

Harmonics: The physical model supports natural harmonics, as shown in Figure 2.10. These are performed by pressing lightly with both the left-hand finger and bow. To perform harmonics with the control loop, a second set of SVMs would need to be trained to recognize the timbre of harmonics. These would then adjust the bow force in order to achieve and maintain harmonics. The physical model does not support artificial harmonics, as this technique requires two fingers to be placed on the string.

Natural harmonics are not introduced until Suzuki book 4, while artificial harmonics are an advanced technique which is not expected from violinists with less than 10 years of experience.

Vibrato: This technique is a sinusoidal modification of finger position of approximately 5.5 Hz for both violin and cello (Geringer & Allen 2004). Modifying x_g is of course supported by the physical model, but the left-hand control loop will need alterations. In particular, rather than adjusting x_g according to the last 3 ticks of output, the average pitch over the past 10–20 ticks (since one cycle of vibrato takes 9 ticks) must be taken in order to smooth out the deliberate alterations in pitch due to vibrato. After determining and adjusting the “average” x_g , the sinusoidal movement is added to form the actual x_g used.

The physical model is faster when finger and bow positions do not change, as any modification to x_f or x_b requires a recalculation of the eigenvectors $\phi_n(x_f)$ or $\phi_n(x_b)$. Such recalculations could be avoided by discretizing finger and bow positions (say, 1000 values for x_f between 1.0

and 0.667 and x_b between 0.05 and 0.2) and pre-computing all eigenvectors for those values. This would produce a table of 312 kb, which will not stress the memory allocation of a modern desktop computer.

Vibrato is usually taught to students with 3–4 years of experience.

Sul tasto: This is bowing over the fingerboard, i.e. $x_b \geq 0.2$. Following the design of existing dynamics in Table 5.1, *pp* would occur in this range. Alternatively, a mark of “sul tasto” could be considered to be a “dynamic modifier”; i.e. *f sul tasto* could mean “ $v_b = 0.4 \text{ m/s}$, $x_b = 0.25$ ”. Once the new “dynamic” was defined, it would be trained in the same manner as other dynamics. This notation is normally introduced after 4–5 years of experience.

Ricochet, jeté, and sautillé: Bow-strokes controlling a bouncing bow are moderately advanced violin techniques wherein gravity and the elasticity of the bow hair aid the violinist in producing a large repeated variation in bow force. There is no support for such bow-strokes in the present system. The physical modelling would need to be extended to include at least one more axis of bow vibration. If the bow string’s length is the x axis and the bow primarily moves in the y direction, the simulated string currently only vibrates in the y direction but motion in the z direction would need to be added. In addition, the bow must be simulated as an elastic, vibrating system.

These “bouncing” bowing styles are learned over many years, but a violinist with 5 years of experience would be expected to demonstrate rudimentary control over them.

Left-hand *pizzicato*, thumb *pizzicato*: These notations indicate a change of style for the plucked strings. Left-hand *pizzicato* occurs with very large pluck-bridge distances, such as $x_p = 0.8$ or $x_f = 0.9$ rather than the usual $x_p = 0.25$ to $x_p = 0.5$. Thumb *pizzicato* has a wider plucking width W_p and altered spring and decay constants K_p and R_p . The difference between thumb and finger *pizzicato* will likely not be captured well with only two plucking finger forces; if a greater variation in *pizzicato* style is desired, then extending the physical modelling to use 3 or more forces for the plucking force would likely be required.

Pizzicato is not a priority for most violin students, but it plays a greater role in cello ensemble music, particularly when playing in a jazz style. Such variation in plucking technique may be encountered between 5 and 10 years of experience.

Scordatura: This technique consists of altering the pitches of strings, for example changing a cello’s strings from ADGC to GDGC as is sometimes done for the performance of Bach’s unaccompanied suite for cello no. 5 in C minor, BWV 1011. At the present time, the pitches of instrument strings are implemented as constants, but the string tensions could be varied and the calculation of finger position could be altered to accommodate scordatura.

Scordatura is not introduced in the first 5 years of experience, and some students may not encounter it until 10 years or later.

Col legno: This is a pair of techniques involving the wooden stick of the bow. Depending on the circumstance, it can either mean to hit the string with wooden part of the bow, or to scrape the string with the wood. Some musicians do not perform *col legno* with their bow for fear of damaging the bow (which can cost thousands of dollars); they use a pencil instead. Neither technique is currently supported by the physical model. Bowed *col legno* could be added by

allowing the friction coefficients μ_s and μ_d to be significantly reduced. Struck *col legno* could be modelled as a very short, very fast-moving pluck.

Col legno is not introduced in any Suzuki books. It is not unusual for a string player to first encounter this technique in orchestral music.

Ponticello: The same considerations as harmonics apply here. The physical model supports low bow-bridge distances (which is the defining characteristic of *ponticello*), but a new set of SVMs would need to be trained to recognize the desired timbre.

Ponticello is generally not used in solo work; like *col legno*, it is normally only encountered in orchestral music.

5.4.4 Mid-level note modifiers

The physical actions required to perform bowed notes varies based on the musical score. The main examination of translating between musical notation and physical actions takes place in Section 7.1, but this section will describe the low-level modifiers which effect the control loop. The public API for notes is given in Figure 5.12. Each note is specified with a `NoteBeginning`, `NoteEnding`, and duration in seconds.

Two parameters apply to both the note beginning and ending, while the rest apply to either the beginning or ending.

`physical [both]`: For most notes, the physical actions are only specified for the beginning of the note. If a note's physical actions other than F_b should change throughout the note (e.g., *crescendo*, *glissando*), the ending physical actions must be specified. The physical actions will be linearly interpolated between the beginning and ending actions.

`midi_target [both]`: This sets the reference pitch for the left-hand control loop. If specified for the note ending, the pitch target will be linearly interpolated throughout the note.

`ignore_finger [begin]`: Normally x_g is set to an initial estimate based on the note in the score, but in the case of a tied note we do not wish to modify x_g .

`keep_bow_force [begin]`: Do not set $F_b(t)$ to F_b^i . This is used for slurred notes on the same string.

`keep_ears [begin]`: Do not reset past history of pitches. This is useful for tied notes.

`bow_position_along [begin]`: Explicitly set the bow-string contact point at a specific distance along the bow hair. This has no effect on the audio and haptic simulation; it only affects the video output. It is triggered by explicit annotations in the score such as “frog” or “tip”.

`lighten_bow_force [end]`: Reduce bow force at the end of the note. This is intended to dampen string vibrations immediately prior to changing strings.

`keep_bow_velocity [end]`: Normally the bow decelerates at the end of the note; this option maintains the bow speed. Useful for tied or slurred notes.

```

struct PhysicalActions {
    int string_number;
    float dynamic;
    float finger_position;
    float bow_bridge_distance;
    float bow_force;
    float bow_velocity;
};

struct NoteBeginning {
    PhysicalActions physical;
    bool ignore_finger;
    bool keep_bow_force;
    bool keep_ears;
    float bow_position_along;
    float midi_target;
};

struct NoteEndings {
    PhysicalActions physical;
    bool lighten_bow_force;
    bool keep_bow_velocity;
    float midi_target;
};

```

Figure 5.12: Low-level note modifiers. By default, all values are disabled (for floats, negative values are interpreted as disabled).

Chapter 6

Calibration, Performance, and Self-Improvement

This chapter

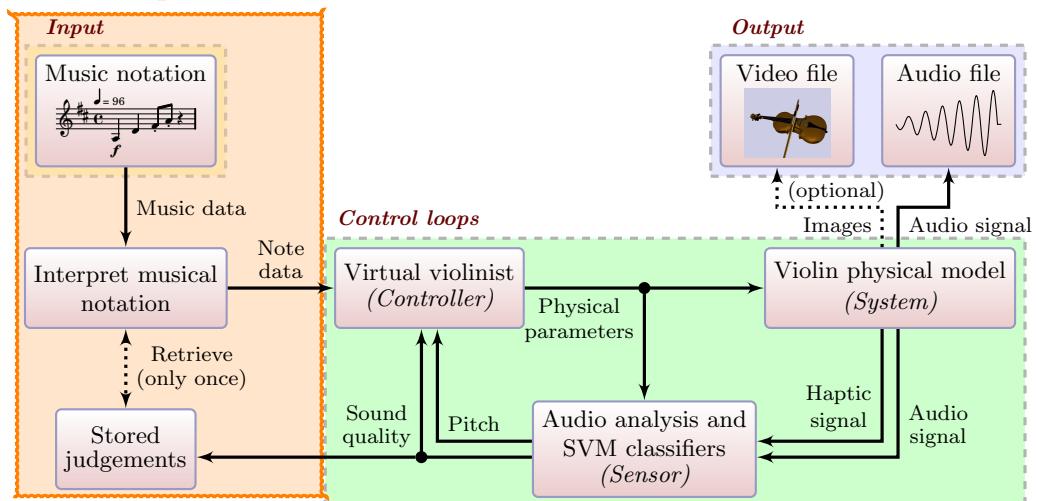


Figure 6.1: Calibration, performance, and self-improvement in context.

The previous chapter discussed the control loops at the center of *Vivi, the Virtual Violinist*. The bowing control loop depends on certain parameters, most notably the initial bow force F_b^i . This chapter discusses the automatic calibration of such parameters, the interpretation and translation of musical scores into mid-level note modifiers, and a method of autonomously “practicing” a piece of music to improve the audio output.

Before the control loops can be used to perform music, the SVM for each string must be checked by the human teacher to ensure that they have adequate coverage for the strings. If any errors are found, the teacher identifies the specific weaknesses and the process begins again. Once this interactive process is completed, the SVMs are used to calibrate the constants needed for the bowing control loop.

Music is performed by extracting musical events from scores, then converting the events into a series of notes as per Section 5.4.4. *Vivi* can autonomously improve the audio output by varying F_b^i for each note over multiple simulations, then selecting the F_b^i which results in the best timbre.

6.1 Training and calibration

Once the basic training (Section 5.2.1) has been completed, a few tests performed by asking the human teacher to listen to a few simulated notes to identify any missing areas in the coverage of the SVMs. After the teacher is satisfied by the performance of the sound timbre judgements on simple notes, three bowing parameters are calibrated. These parameters are then tested by performing a few musical exercises, again with human listening and making interactive corrections if necessary.

6.1.1 Interactive training

All of the output performed by *Vivi* can be presented to the human teacher for closer examination and optional retraining. Figures 6.2 and 6.3 show this process: Part of the audio is selected. If the teacher indicates a timbre class judgement (from 1 to 5, as per Table 5.2), that portion of the audio will be extracted, along with the corresponding portion of the haptic forces and the violinist actions. These signals are added to the dataset in preparation for retraining the relevant SVM.

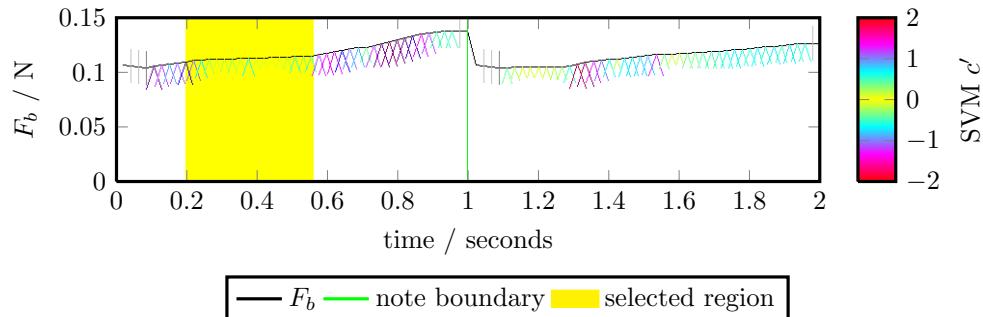


Figure 6.2: Interactive training with a calibration exercise, two notes shown. The size and colour of arrows indicate the magnitude of SVM output c' , while the direction of arrows (up or down) indicates whether $c' > 0$ or $c' < 0$. The teacher can listen to the entire file, or select a portion of the note. The graph and arrows are a screenshot from *Vivi*. In this example, the teacher has selected the region from 0.2 to 0.55 seconds with the computer mouse. The violin sound is “wispy”, so the SVM judgements c' in this region should be -1, not close to 0. The beginning of the second note contains similar errors: The audio sounds wispy but the SVM output c' is close to 0. After the teacher labels these regions, the SVM will be retrained and the exercise will be performed again.

Audio 6.1: Interactive training of a violin calibration exercise

<http://percival-music.ca/dissertation/a.6.1.training-violin-g-f-6.wav>

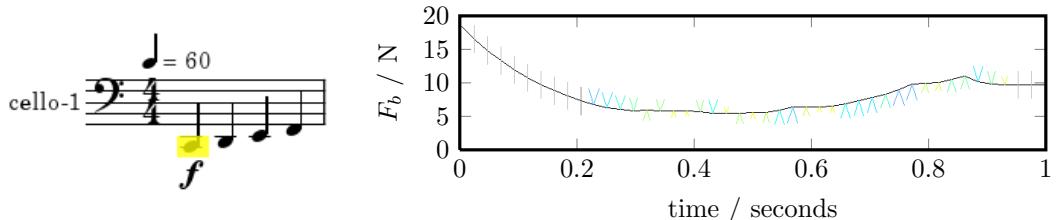


Figure 6.3: Interactive training with cello sheet music, single note shown. Music notation and graph are screenshots from *Vivi*. The teacher can listen to the entire score, or select an individual note as was done in this example. Once a note is chosen, a region within that note can be selected and labelled if corrections are necessary.

Audio 6.2: Interactive training with cello sheet music

<http://percival-music.ca/dissertation/a.6.2.training-cello-c-f.wav>

6.1.2 Accuracy of the SVMs

After training the SVMs, two quick checks of their accuracy is computed:

10-fold cross-validation: This is the typical measure of accuracy used in Music Information Retrieval. The training dataset is randomly split into 10 subsets of equal size. The SVM is trained on 9 of those subsets, then tested on the remaining subset. This process is repeated 10 times, using a different subset as the testing set each time.

Testing individual files: The SVM is trained on the entire dataset, then is used to predict the sound quality judgements for each file. This is not an accepted method of testing the accuracy of a dataset, since the training set is equal to the testing set. However, the purpose of this check is not to find the overall accuracy, but rather to catch any major errors in the labelling. There are two types of errors which can be discovered at this stage. First, an audio file may be mislabelled in its entirety. This usually occurs due to human error (e.g., pressing the wrong key by accident), but occasionally occurs from the user's opinion of a particular file being revised (e.g., a "wavy" sound may be judged as category 1 on one day and category 2 on another day). Second, the audio file may contain changing timbre. This occurs most often when attempting to label audio during a note attack; the first half of the audio may be category 2 or 4 (poor timbre) while the second half of the audio may be category 3 (good timbre). In these cases, the user should label sub-portions of the audio file accordingly, then delete the combined (mislabelled) file.

On a practical note, it is rare to find an error in the data labelling, but the tests are worth performing since they only require a few seconds to compute. The 10-fold cross-validation accuracy of each string was between 98.8% and 99.9%.

6.1.3 Verifying the SVMs

Checking for mislabelled data is no guarantee that the basic training in Section 5.2.1 will have encompassed the entire range of sounds which arise while performing music with that instrument. An SVM can be 100% accurate (with cross-validation) while failing to adequately control sounds during simulation which are quite dissimilar to the sounds encountered during the steady-state basic training. There is a trade-off between the number of audio files to label and the amount of coverage.



Figure 6.4: Checking the output of the SVMs, screenshot from *Vivi*. Each cell is a plot like Figure 6.2 without the axes, and the vertical axis (F_b) is unique for each cell. In addition to displaying the force profile and SVM judgements, each cell shows the mean c' over the two notes. Rows indicate "basic" finger positions (open string, one semitone, and six semitones). Columns are sorted from lowest to highest force.

The extreme judgements (red) are expected in the outer left and right columns; the touch of green in the upper-left cell is suspicious and probably contain misjudged audio. The teacher would then select that cell, listen to the audio, and label the incorrect material.

To check that the SVMs are useful, *Vivi* generates a small number of examples, arbitrarily chosen to be 9 per basic finger. Each example consists of two notes: a downbow and upbow. One set is shown in Figure 6.4. All examples use $K = 1.01$ (i.e. very gradual adjustment) as this slows the variation of string timbre, making it easier to select and retrain any questionable judgements. While the primary goal of this exercise is to verify the SVMs, the secondary goal is to explore the range of F_b^i which may produce good output.

The “verifying SVM” exercise is conducted as follows. For each file, the mean of the judgements $c'_m = \frac{1}{n} \sum_{i=1}^n c'[n]$ of the two notes is considered to be the overall judgement for that note. This is a fairly crude measure since the judgements often change over time, but it is sufficient for this purpose.

1. Exercises are generated by setting F_b^i to the minimum and maximum bow forces found in the dataset. The mean of bow force judgements c'_m is computed, and should be close to -2 and 2 (the two extreme values). If this is not the case, a warning is displayed. The initial bow force F_b^i of each file is saved in a set of V_l (low) and V_h (high), respectively.
2. The next exercise is generated with F_b^i being the mean of the maximum V_l force and the minimum V_h force. If the mean c'_m of judgements of this file is lower than -0.25, it is appended to V_l ; if it is higher than 1.0, it is appended to V_h . If the mean is between -0.25 and 1.0, it is saved to a new V_m (middle) set. The upper and lower bounds on the means differ because the bow control loop assumes that F_b^i will be higher than the steady-state F_b (Section 5.2.3).

This step is repeated until V_m is non-empty.

3. The final task is to reduce the distance in F_b^i from V_l to V_m , and V_m to V_h . To pick which region to examine, two ratios are computed: $\frac{\min(V_m)}{\max(V_l)}$ and $\frac{\min(V_h)}{\max(V_m)}$, indicating the range between the low, middle, and high sets. The next exercise is computed with F_b^i being in the middle of the largest ratio. Just as with step 2, this new exercise is saved to V_l , V_m , or V_h depending on the mean of bow force judgements c'_m .

This step is repeated until there are a total of 9 files.

Once these steps are completed, the minimum and maximum F_b^i in V_m are stored for future use. I will refer to these as V_m^- and V_m^+ . When training a new string, I perform this exercise approximately five times, adding another 15–20 labelled audio files to the 48–66 files labelled during the basic training.

6.1.4 Initial bow force F_b^i and correction factor K

Although I employed a search algorithm for an initial exploration of F_b^i in the previous section, it was not a very robust method since the sound produced by a particular F_b^i can vary by a factor of 2 depending on the value of K . If I could be guaranteed a good value of either F_b^i or K , then I could do a one-dimensional search for the other variable. However, lacking a reliable estimate of either variable, I treated this as a two-dimensional optimization problem and solved it with a grid search.

The bow force estimates V_m^- and V_m^+ offer a very narrow range of initial bow forces due to the small value of K used. To test a larger range, I examine nine forces spaced linearly between V_m^- and $2V_m^+$. The upper bound V_m^+ was increased because the bow control loop automatically reduces the bow force and the verifying stage only used a small value of K . The factor of 2 was found experimentally to cover a good range on all strings from cello C to violin E. 9 values of K are also evaluated, spaced



Figure 6.5: Simulated notes for bow force calibration. The light double bar lines indicate that the physical model is reset. This evaluates the suitability of F_b^i when starting from a stationary string and already-vibrating string.

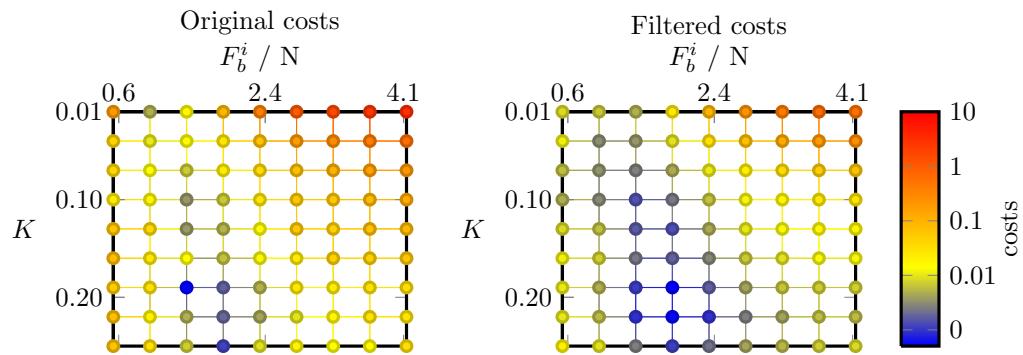
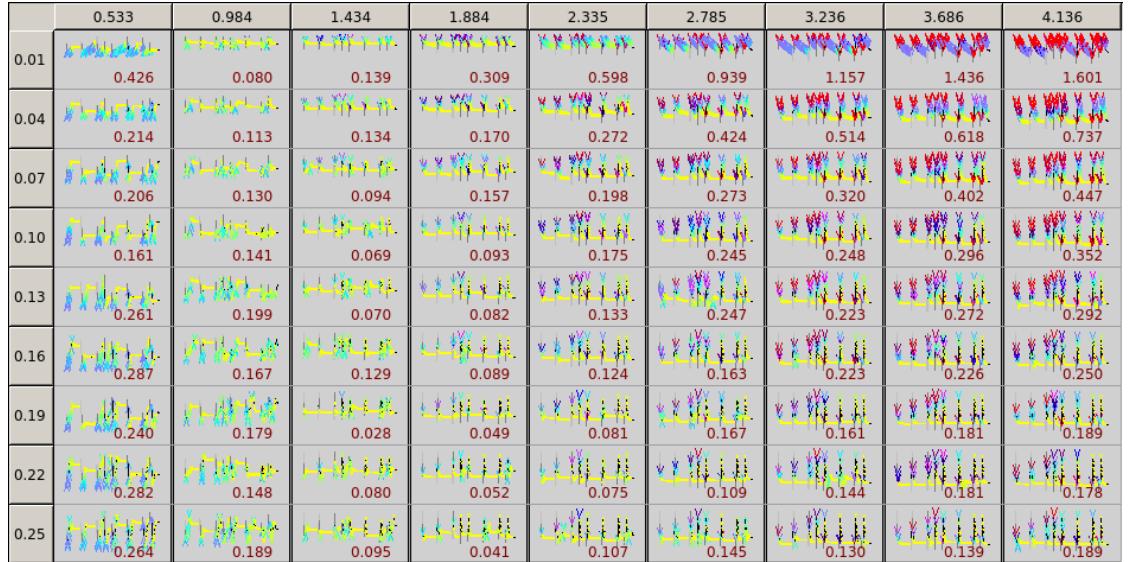


Figure 6.6: Calibrating F_b^i and K , violin open D string. Top: screenshot from *Vivi*, each cell contains four notes. Users may click on each cell for additional information.

Bottom: the costs of each cell, with the original values (bottom-left) being the costs shown in the top screenshot. This display format is recommended for non-interactive media (i.e. this dissertation rather than the *Vivi* software).

linearly between 0.01 and 0.25. For each pair of variables, four pairs of two notes were generated in order to account for randomness in the string behaviour in the pattern shown in Figure 6.5.

Since this is a more thorough examination of F_b^i and K , the cost metric for each note is more sophisticated than the mean c'_m utilized in the previous section. The list of judgements C for each file is split into separate notes C_1, C_2, C_3, C_4 , and the total cost is

$$\text{cost of } F_b^i, K = \frac{K}{|C|} \sum_{n=1}^4 \sum_{i=1}^{|C_n|} t \cdot |C_n(t)| \quad (6.1)$$

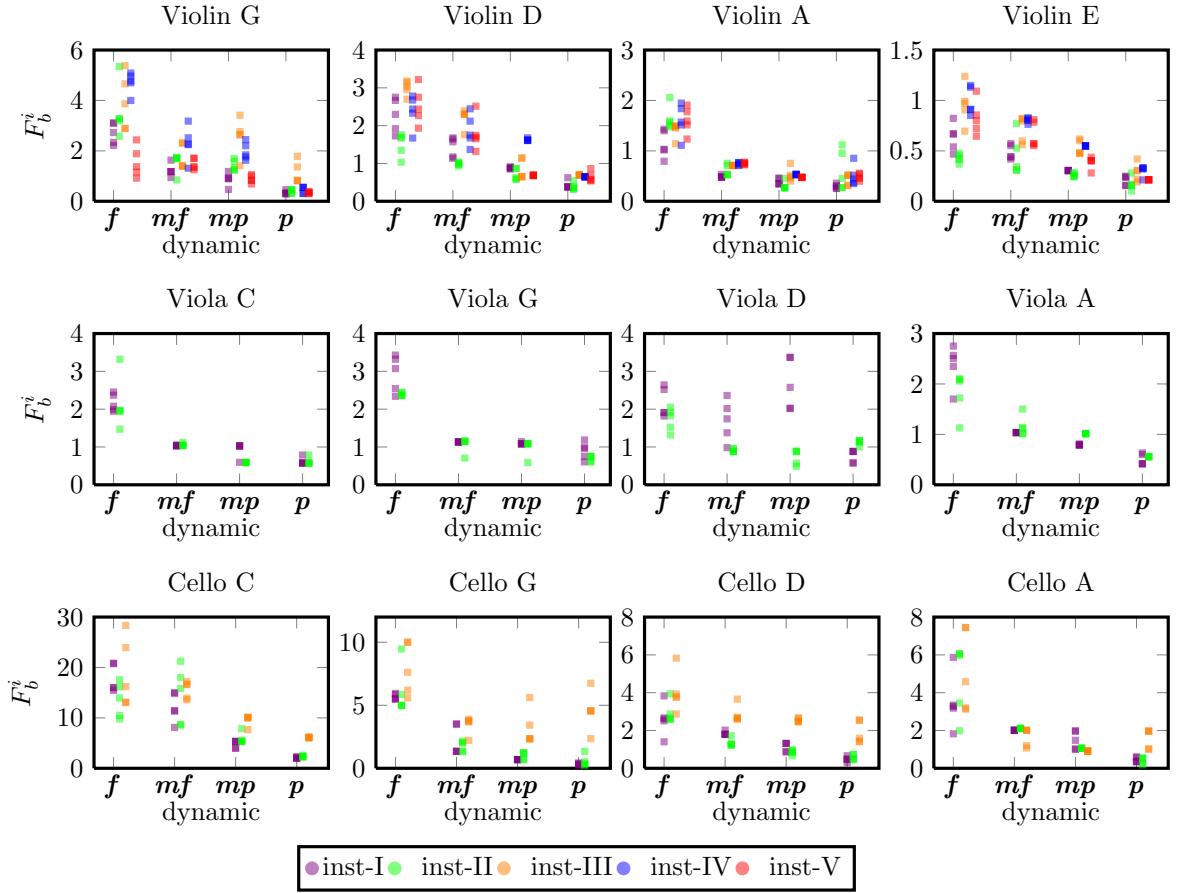


Figure 6.7: Calibrated F_b^i , five experiments performed for each instrument. K factors not shown in this plot. Note that the y-axis differs between plots.

As expected, F_b^i decreases for higher-pitched strings and for quieter dynamics. The calibrated F_b^i for most strings are fairly close, with a few notable exceptions: Viola-II D mf and mp , and all Cellos A f . The spread of F_b^i between different instruments (such as Violin-I vs. Violin-V) is not a problem; we should expect strings with different physical constants to require different amounts of force.

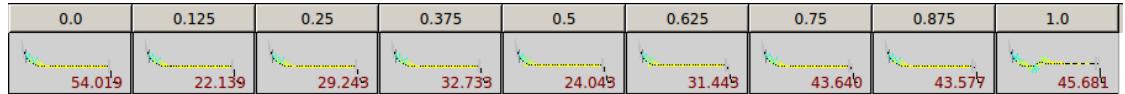
In other words, the penalty of each tick is the absolute value of the bow force correction c' , linearly weighted by its position within the note. The intent is to penalize corrections which occur later within a note. Furthermore, the overall cost is penalized by multiplying it by K : If all else is equal, then a gradual change of bow force will produce more reliable bowing than large changes.

An example of these files and their costs is shown in Figure 6.6. Close examination of that figure shows that the cost of individual notes varies somewhat; in order to avoid a random local minimum, the costs are smoothed by applying a low-pass filter in a cross pattern. In particular, the filtered cost $C'[i][j]$ of each cell $C[i][j]$ is given by

$$C'[i][j] = \frac{C[i][j]}{2} + \frac{C[i][j-1] + C[i][j+1] + C[i-1][j] + C[i+1][j]}{8} \quad (6.2)$$

For cells on the edges, the cells outside the boundaries are removed from the sum, and the denominator is reduced by two for each missing cell.

The minimum filtered cost C' indicates the values of F_b^i and K which are selected. This process is repeated once for each “basic” finger position. A set of calibrated bow forces is shown in Figure 6.7.

Figure 6.8: Calibrating bow lifting factor D_b , violin open D string, screenshot from *Vivi*.

6.1.5 Bow lifting factor D_b

When the bow changes from one string to the next, the first string will still be vibrating while the second string is played. This is physically accurate, but is generally not musically desirable unless it is specially notated (“l.v.”, for *laissez vibrer*). I therefore calibrate a bow lifting factor, D_b .

The goal of damping the string is to reduce the audio output as much as possible. Due to the lack of finger damping, open strings vibrate much longer than fingered notes, so only the open strings are evaluated. Nine values of D_b are tested, spaced linearly between 0 and 1. For each file, the string is bowed for 1 second of simulated time using the F_b^i and K chosen in the previous section, with the bow force being reduced with a candidate D_b as described in Section 5.2.3. Following that, the string is left to vibrate for 1 second of simulated time. The cost is simply the RMS of the audio output for the time after the bow is removed from the string.

$$\text{cost of } D_b = \sqrt{\frac{1}{|A|} \sum_{t=1}^{|A|} A[t]^2} \quad (6.3)$$

An example of this is shown in Figure 6.8.

6.1.6 Musical exercises

Once the previous steps are completed, *Vivi* is almost ready to perform real music. So far all simulations have occurred using the “basic” finger positions, with almost all audio being labelled using the first instrument of each type (i.e. violin-I, viola-I, cello-I). The SVMs generally require another 5–10 labelled examples with other finger positions, and a further 2–5 exercises for each instrument to avoid being confused by the different instrument body impulse responses. Once that training is completed, there is the additional problem of *crescendo* and *decrescendo*: The SVMs must be able to produce good judgements when the bow position and velocity are linearly interpolated between the pre-set dynamics. The final number of labelled files in each dataset is shown in Figure 6.1, and the full datasets are available as discussed in Appendix C.

This additional interactive training could be performed using any sheet music, but I prefer to use musical scales and exercises shown in Figure 6.9. The use of scales arises partly from music pedagogy, where they are traditionally a fundamental step in instrumental practice; as a violin teacher, it “feels

String	Files	String	Files	String	Files
Violin G	138	Viola C	64	Cello C	102
Violin D	115	Viola G	74	Cello G	76
Violin A	122	Viola D	68	Cello D	79
Violin E	110	Viola A	77	Cello A	80

Table 6.1: Number of files labelled in the datasets. The number of examples in the dataset varies based on the amount of audio which was labelled while verifying the SVMs and performing these musical exercises.



Figure 6.9: Musical exercises for *Vivi*. The two scales are also performed with *mf*, *mp*, and *p* but are not shown here. The scale with separate notes tests a new F_b^i for each note. The scale with slurred notes only uses a new F_b^i for the first note on each string; as such, errors in the SVM output causing improper F_b will tend to accumulate, which is useful for identifying such problems. The exercise with *crescendo* and *decrescendo* practices varying the dynamic within each note.

Audio 6.3: Violin musical exercises

```
http://percival-music.ca/dissertation/a.6.3.violin-scale-separate.wav
http://percival-music.ca/dissertation/a.6.3.violin-scale-slur.wav
http://percival-music.ca/dissertation/a.6.3.violin-cresc-decresc.wav
```

Audio 6.4: Viola musical exercises

```
http://percival-music.ca/dissertation/a.6.4.viola-scale-separate.wav
http://percival-music.ca/dissertation/a.6.4.viola-scale-slur.wav
http://percival-music.ca/dissertation/a.6.4.viola-cresc-decresc.wav
```

Audio 6.5: Cello musical exercises

```
http://percival-music.ca/dissertation/a.6.5.cello-scale-separate.wav
http://percival-music.ca/dissertation/a.6.5.cello-scale-slur.wav
http://percival-music.ca/dissertation/a.6.5.cello-cresc-decresc.wav
```

natural” to hear (and correct) mistakes in timbre in scales. However, there are also solid technical reasons to practice scales before attempting more complicated music: It allows us to pinpoint and fix problems in isolation. Correcting mistakes in musical exercises involves a certain amount of repetition — identify a few mistakes, label them, retrain the SVM, then perform the exercise again. Retraining the SVM takes approximately five seconds, and simulating any of the exercises shown in Figure 6.9 takes less than one second. Longer music (naturally) takes more time to simulate, and longer to listen to. The musical exercises cover most of the potential problems which would be found in real music, so it is appropriate to ensure that the performance of these exercises is satisfactory before attempting more complicated sheet music.

6.2 Interpretation of musical notation

Vivi reads music scores in the LilyPond format, a text-based system for engraving sheet music. An example is shown in Figure 6.10. Technical details of extracting musical events from LilyPond scores are given in Section 7.1. The term “music events” includes notes, rests, dynamics, articulations, slurs, fingerings, etc.

Once a list of music events has been obtained, they must be mapped onto the mid-level note modifiers described in Section 5.4.4. I begin by discussing the handling of notes without any special modifiers. The remaining musical notation is split into two categories: Notation for which there is no uncertainty in terms of how the note(s) should be modified, and notation which has no fixed meaning. The latter category is known amongst musicians as “interpretation”, and is a matter of research and debate about historical performance practices, the composer’s intentions, and/or the performer’s desires. It would be very interesting to examine the subject of teaching an artificial intelligence to produce musical interpretations, but that is beyond the scope of this research. For the purpose of this dissertation, I will hard-code a particular interpretation for all “ambiguous” notation, choosing values which mimic the behaviour of beginning Suzuki students.

6.2.1 Default note handling and pitch

Each note must have at least five parameters specified: The string number s , finger position x_g , bowbridge distance x_b , bow velocity v_b , and initial bow force F_b^i . The reference pitch P_r is useful but not strictly necessary; if it is omitted then the left-hand pitch feedback loop will be disabled.

Unless otherwise specified, *Vivi* assumes that each note is played on the highest string s which can produce the desired pitch; in more musical terms, *Vivi* assumes that all notes are in first position, other than higher-position notes on the E string. If the musical score indicates that notes should be played on a particular string such as shown in Figure 6.12, that notation overrides the default behaviour.

```
\relative c' {
  \key d \major
  \tempo 4 = 96
  a4\f d fis8-. a-. r4 |
  d16(\downbow cis b a) g4
  \breathe e8\p( g) fis4 |
}
```

The generated musical score consists of three bars. The first bar starts with a forte dynamic **f** and contains four notes: G, D, A, E. The second bar starts with a dynamic **f** and contains four notes: G, D, A, E. The third bar starts with a dynamic **p** and contains four notes: G, D, A, E. The tempo is set to 96 BPM.

Figure 6.10: Sample of LilyPond input format with generated output.



Figure 6.11: String and finger positions. The first bar contains all open strings, G D A E from bottom to top. The second bar is all on the G string since all the notes are lower than the open D string. The third bar is also played on the G string, but only due to the additional notation: The 4th finger, the IV marking on the individual note, and the IV with the extender line indicating that all notes under it should be played on the G string.

The finger position x_g and reference pitch P_r are set according to equal temperament. Given a note with a MIDI pitch value of m ,

$$P_r = 440 \cdot 2^{\frac{m-69}{12}} \quad (6.4)$$

If a different tuning system was desired (e.g., Pythagorean, Just intonation, 19-tone equal temperament), (6.4) could easily be modified so that *Vivi* would play in that style. It would even be possible to allow *Vivi* to select different tuning systems automatically depending on the instrumentation of the work — violinists playing by themselves or in groups often use Just intonation, whereas violinists playing with a piano will use equal temperament. However, at the moment *Vivi* only supports twelve-tone equal temperament.

The bow-bridge distance x_b and bow velocity v_b are set according to the values in Table 5.1. Unless there is a slur notated in the music (discussed later), the bow velocity v_b alternates every note by multiplying it by -1. If there is no bow direction or dynamic specified, *Vivi* assumes that the score begins with a downbow ($v_b > 0$) and **f**.

If the pitch is one of the “basic finger” positions, then F_b^i and K are set according to the values determined in Section 6.1.4. If the pitch is between 1 and 6 semitones above an open string, the values of F_b^i and K are linearly interpolated between the calibrated values for 1 and 6. Finger positions higher than 6 semitones will have F_b^i and K set as the value calibrated for 6 semitones.

6.2.2 Notation with fixed meaning

Notation with an unambiguous interpretation is shown in Figure 6.12.

Slurs, ties, and bow direction

When two or more notes are connected with a slur, the bow direction does not change between notes. In particular, v_b does not alternate, and the interior notes have `noteBeginning.keep_bow_force` and `noteEnding.keep_bow_velocity` set to true. For the two ends of the slur, the beginning note of the slur only enables `noteEnding.keep_bow_velocity`, while the ending note of the slur only enables `noteBeginning.keep_bow_force`. Two notes connected with a tie receive the same treatment as notes connected with a slur, with two additional modifications: The final note of the tie enables `noteEnding.ignore_finger` and `noteEnding.keep_ears`.

The bow direction can be specified with downbow and upbow signs. These set v_b as indicated, but do not enable `keep_bow_force` or `keep_bow_velocity`. In other words, the bow velocity returns to 0 at the end of each note, and each note begins with a new F_b^i .

Pizzicato and *arco*

An indication of *pizz.* (short for *pizzicato*) indicates that all following notes should be plucked as per Section 5.4.1. An indication of *arco* indicates that normal bowing should resume.



Figure 6.12: Slurs, ties, *pizzicato*, and *arco*.

6.2.3 Notation with ambiguous meaning

Notation open to interpretation is shown in Figure 6.13.

Dynamics

As discussed in Section 5.2, dynamics set x_b and v_b to the somewhat-arbitrary values specified in Table 5.1. Hard-coding the effects of dynamics in this manner is not ideal, but it suffices for beginning-violinist audio. Gradual changes of dynamics (i.e. a *crescendo* and *decrescendo*) are assumed to be a linear interpolation of x_b and v_b between the starting and ending dynamic. Again, this does not reflect the actual behaviour of skilled violinists, but this ad hoc behaviour is adequate for *Vivi*'s current skill level.

Articulation

Staccato and *portato* articulations indicate that the note should be shorter, but their exact musical meaning varies based on historical period, composer, and even the musician's personal style. I arbitrarily set a *staccato* note's duration to be 0.7 times the original duration (followed by a rest which is 0.3 times the original duration); a *portato* note is 0.9 times the original duration (again followed by a rest). When these articulations occur within a slur, they behave similar to two downbows in a row — the bow direction does not change but the bow velocity decelerates to 0 at the end of each note.

A breath mark sets the previous note to 0.5 times the original duration (followed by a rest). In addition, the note has the additional modifier `noteEnding.keep_bow_velocity`. An accent indicates that the note should be “stronger”. In *Vivi*, this is interpreted as doubling that note's F_b^i .

Lighten bow and position along bow hair

If the bow pressure is released suddenly (as a result of changing strings or moving to a rest), it creates an undesirable pluck-like sound. To lessen this effect, `noteEnding.lighten_bow_force` is enabled in these cases, which then uses the bow lifting factor D_b calibrated in Section 6.1.5. There is one exception: If the final note has a breath mark, then `noteEnding.lighten_bow_force` is not enabled, as it is desirable for the string to ring for a longer time.

On a real violin, playing closer to the frog results in a heavier bow (increased F_b), while playing closer to the tip results in a lighter bow. However, this aspect of violin mechanics is not included in the physical model, so these indications are purely cosmetic for the video generation. The notation “frog” sets the bow-contact point along the bow hair to 0.0; “lh” (lower half) is arbitrarily 0.2, “mid” is 0.5, “uh” (upper half) is 0.8, and “tip” is 1.0.

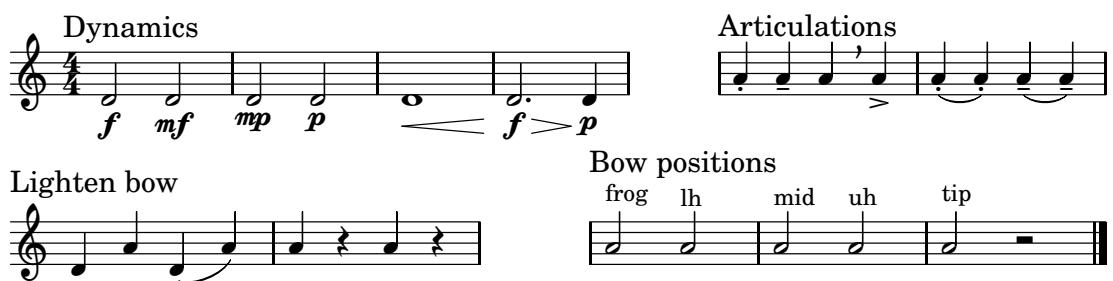


Figure 6.13: Musical notation with no fixed meaning.

6.3 Practicing a piece of music

As mentioned in Section 6.1.1, all output produced by *Vivi* can be examined and problems can be identified for retraining. This section discusses autonomous improvements (i.e. without human listening or interaction) to the music performed by *Vivi*.

The values of F_b^i selected in Section 6.1.4 are an imperfect prediction of the appropriate bow force. There are two problems with these estimates: First, the grid of tested bow forces is fairly coarse; second, those estimates do not take into account the previous note(s) played on the string. The actions required to produce a good tone on a string at rest differ from the actions required to produce a good tone on a string which is already vibrating. Even if we increased the resolution of the F_b^i candidates and calibrated F_b^i for every finger position (rather than only evaluating the “basic” finger positions), the final F_b^i figures would not be suitable for all string states.

To resolve this, I again take inspiration from another aspect of violin pedagogy: Solo practice. Music students play a piece of music dozens or even hundreds of times before performing the music in public. While playing the music alone, the student listens to the sound and adjusts the bowing parameters. If a particular mistake is made in the same place multiple times, the student should remember to adjust bowing parameters in advance to avoid those mistakes. The student may even add an annotation to the musical score as a reminder.

To imitate “solo practice”, *Vivi* will use a simple (yet effective) algorithm: Hill climbing (Minsky 1961). Imagine standing at one point in a three-dimensional surface, then look at the elevation one step away in all directions. Take one step in the direction which results in the highest elevation. Repeat until you cannot find a higher elevation within one step. This is an iterative local search anytime algorithm: The longer it runs, the better the outcome will be, but if it is interrupted after a short time it will still return a valid result. It is subject to converging on a local maxima rather than a global maxima, but it produces adequate results for my problem space. In the case of *Vivi*, the “steps” are variations in F_b^i for each note. There are two steps to this process: Generating the candidates to examine, then choosing a new set of F_b^i .

6.3.1 Generating hill-climbing candidates

I test different values of F_b^i for each note of the score. To reduce the chance of getting stuck at a local minima, I generate five different candidates, setting $F_b^i \leftarrow hF_b^i$ with

$$h = \left\{ 1.0, \quad \mathcal{U}(1.05, 1.15), \quad \frac{1}{\mathcal{U}(1.05, 1.15)}, \quad \mathcal{U}(1.2, 1.4), \quad \frac{1}{\mathcal{U}(1.2, 1.4)} \right\} \quad (6.5)$$

where $\mathcal{U}(a, b)$ is the uniform random distribution between a and b . Each candidate h is applied to all notes. The self-improvement could be more effective if all combinations of h were tested for each note, but this would result in 5^N combinations, where N is the number of notes in the score. Even a simple one-octave scale (even simpler than the exercises in Figure 6.9) contains 8 notes, meaning 390625 combinations, well beyond the bounds of practicality. This could be reduced to 2^N if the set of choices was reduced to “reduce bow force, or keep it the same”, but the 8-note scale would still require 256 combinations. I therefore apply each h to all notes in the score. As noted previously, the stochastic string behaviour requires that multiple experiments be performed. I therefore simulate the entire piece of music 20 times for each candidate h , for a total of 100 performances.

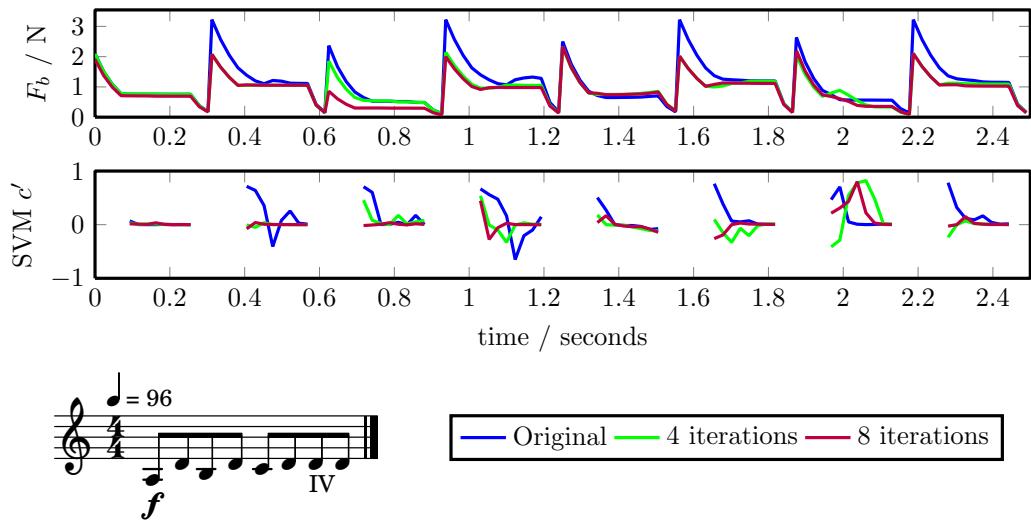


Figure 6.14: Practice with hill climbing. Note that the improvement varies quite a bit for each note: The first note shows no change throughout the iterations, while the second note shows dramatic improvement. The second-last note — the D played on the G string — does not show any real improvement.

Audio 6.6: Examples of solo practice

<http://percival-music.ca/dissertation/a.6.6.violin-hill-0.wav>
<http://percival-music.ca/dissertation/a.6.6.violin-hill-4.wav>
<http://percival-music.ca/dissertation/a.6.6.violin-hill-8.wav>

6.3.2 Choosing the steps

After the sheet music has been performed with the variations in F_b^i , I evaluate each note. The cost of each note o is the sum of squares¹ of bow judgements C_{oh} ,

$$\text{cost of note } o, \text{ adjustment } h = \sum_{t=1}^N C_{oh}[t]^2 \quad (6.6)$$

This results in a list of 20 costs for each h for each note. To pick which h to use for each note, I use the Kruskal-Wallis one-way analysis of variance test (Kruskal & Wallis 1952) to compare each pair of lists. This is the non-parametric version of ANOVA (analysis of variance); Kruskal-Wallis does not assume that the underlying distributions are Gaussian. Provided that the sample size is greater than 5, the test results approximately follow a chi-squared distribution; this is used to generate a p -value.

For any pair of h values for a particular note, I accept the new value of h if $p < 0.01$. That value is saved to a list of alterations a_o for each notes in the score; future performances of that sheet music automatically multiply that note's F_b^i by a_o . If the hill climbing is performed again, then the candidates h are multiplied by a_o for each note.

It may be noted that a new value of a_o for a note at the beginning of the score may cause later notes to adjust to this new a_o . This cascade is an unfortunate but acceptable consequence of this method of practicing a piece of music.

I found diminishing returns after about 5 iterations, and virtually no benefit to repeating this algorithm more than 10 times. One example of this practicing is shown in Figure 6.14.

¹Since these costs are evaluated relative to each other, the rankings will be the same whether sum of square or RMS is used.

6.4 Final remarks on calibration, music, and self-improvement

This chapter discussed the interactive training and automatic calibration of *Vivi*, the performance of musical scores, and a method of improving those performances without human interaction. I end with remarks about potential improvements to *Vivi*'s autonomous functions and to the musicality of performance.

The major research contributions of this chapter are:

- A method of automatically calibrating parameters of a virtual musician based on a human-trained model of evaluating timbre.
- A set of rules to decompose a musical score into a list of discrete notes with no loss of musical information.
- Applying the artificial intelligence technique of hill climbing to the choice of initial bow forces, resulting in “smoother”, more accomplished performances of sheet music after practicing.

There are many alternative approaches to the material in chapter which other researchers could investigate. For example, the cost function for a particular pair of (F_b^i, K) gives greater weight to later ticks within the note. This weighting was chosen as it seemed to produce better audio in informal experiments, but there is no concrete evidence to suggest that it is better than other weightings. On a more general note, the calibration of parameters could be generated in a completely different manner — for example, by measuring the actions of human musicians, estimating their actions through analysis of audio recordings, or by asking musicians to select parameters manually. I chose to limit human involvement to the single action of providing judgements about timbre in order to lessen the amount of skill needed by the human trainers, but it is not hard to hire competent musicians so this may seem unduly restrictive to other researchers.

As noted in Section 6.2, the interpretation of notation with “fixed meaning” is clear, but the interpretation of notation with “ambiguous meaning” was fairly ad-hoc. I manually chose values which produced decent performances of the mostly-Baroque music in the corpus of sheet music. One alternative approach would be to survey musicians in an attempt to find some “common ground” interpretations, but I think that an improved method would be to leave such decisions up to each user herself — these elements of notation are ambiguous precisely because there are multiple valid interpretations. The ideal case would be to allow the composer and/or performer (in this case, the human user) to make such subjective decisions on a case-by-case basis.

6.4.1 Improvements to the calibration and practicing

Given the importance of F_b^i and K to the sound quality, it would seem desirable to improve the resolution of these estimates. However, experiments with a two-stage grid search yielded no perceptual improvement to the music. The first stage of the search functioned as the existing grid search, but the best F_b^i and K are identified, a second search was performed in the surrounding area. For example, for the violin open D string shown in Figure 6.6, the first stage identified $F_b^i = 2.2, K = 1.19$ as the best sound. The second stage then tested ranges related to the initial estimate, namely F_b^i from 1.1 to 4.4 and K from 1.09 to 1.29. The problem is the variation in string behaviour. For any particular F_b^i , there is a probability p_g of having a good bow-stroke, probability p_a of having an almost-acceptable

bow-stroke, and a probability p_b of having a bad bow-stroke. If there is only a small difference between (p_g and p_b) or (p_b and p_c), then many notes must be performed in order to establish good estimates of p_g , p_a , and p_b .

A different strategy would be to differentiate between an F_b^{is} for use when starting a bow-stroke from rest and an F_b^{ic} for use when continuing a bow-stroke on an already-vibrating string. When examining simulations of repeated notes, there is a clear difference in the force required in these two cases. This would slightly complicate the bow control loop — when should F_b^{is} be used, and when should F_b^{ic} be used? A first approximation would be to use F_b^{ic} when the previous note was on the same string; otherwise use F_b^{is} . This would result in F_b^{is} mistakenly being used when alternating quickly between strings (such as the music Figure 6.14), but it is a serviceable first approximation.

Alternatively, a more advanced search technique could examine the beginning of each note and adjust F_b^i and K accordingly. For example, if the first few sound quality judgements c' are positive, then F_b^i would be decreased and/or K would be increased. There are two difficulties with this approach which prompted me to refrain from investigating them in great detail: First, there is no ad hoc reason to prefer modifying F_b^i instead of K (or both at the same time); and second, this relies on accurate sound quality judgements during the note attack. The first problem could be resolved by simply hard-coding a particular value of K . Informal experiments suggest that fixing K does not cause any obvious degradation of sound quality after choosing new values of F_b^i to account for the fixed K . Reducing this problem to a single dimension would greatly simplify the search. The second problem is harder to solve; as discussed in Section 5.4.2, the judgements during the note attack are not as reliable as judgements during the steady portion of the note, particularly for the cello C string.

A similar argument can be applied to *Vivi*'s autonomous practicing of a piece of music: Rather than a brute force evaluation of all pre-determined modifications, a limited set of candidates based on analysis of previous performances could be tested. In particular, rather than blindly modifying all notes, *Vivi* could pick a few bad notes, perform the music many times while only modifying parameters of those specific notes, then evaluate which parameter(s) resulted in improvement(s). This would be a much closer match to the actions of real music students.

Another useful step would be to use the practicing of a single piece of music to improve the overall behaviour of *Vivi*. In the current implementation, the only “long-term” memory of *Vivi* is the annotated datasets used to train the SVMs. *Vivi* uses “short-term” memory to improve the performance of a single piece of music, but no knowledge of note alterations a_o for one piece of music are applied to other pieces. An improvement would be to add a second layer of machine learning which is trained on the musical score and note alterations a_o that arose from of practicing. This second layer of machine learning would then predict a set of a_o for any new scores. For example, suppose that *Vivi* must always increase the bow force of the third finger on the D string. At present, *Vivi* must relearn that by practicing each piece of music. If this feature was added, then after practicing a few pieces of music, *Vivi* could learn that the bow force must always be increased for notes of that pitch.

6.4.2 Improvements to the musical interpretation

The naive hard-coded approach to musical interpretation is sufficient for beginner violin music. Slightly more advanced music would benefit from interpreting a few more symbols.

Trill: This splits a long note into quickly alternating notes; one which is the notated pitch, and another which is one or two semitones above the lower pitch. There are a few stylistic decisions to make: whether the trill should start on the upper note or the lower note, whether the first or last note should be held longer than the other notes, and the exact pitch of the upper note. The latter is not directly a stylistic question — unless specifically noted otherwise, the upper pitch should be the next note in the scale. However, algorithmically determining the scale (and key) of a musical score is still an open problem in music information retrieval.

An acceptable initial approximation would be always start from the initial note, perform the trill with 32nd notes (or “demisemiquavers”), and have the upper pitch be two semitones above the lower pitch. However, there must be a way for the user to override this decision; if a trill occurred on the leading tone of a major scale (a relatively common occurrence), a two-semitone trill would be extremely bad (e.g., trill B–C♯ in C major).

Mordent: This ornament indicates one alternation of a trill at the beginning of a note, and can thus be implemented in much the same manner as a trill.

Turn: This ornament indicates one alternation of a trill at the ending of a note.

Fermata: When placed over a note, this indicates a longer duration; when placed on a barline, this indicates an added rest. The exact lengthening is a stylistic choice; for the initial hard-coding, multiplying the existing duration by 50% or adding a half note rest is sufficient.

The musical interpretation would clearly benefit from allowing the user to tweak the hard-coded values. At the moment this can only be done by modifying constants in the python code, but two much improved systems can be envisioned. First, the user could add annotations to the LilyPond input file to override the default behaviour of *Vivi*, such as `\vivi #'(staccato 'duration 0.6)`. This would require a certain amount of familiarity with LilyPond, but no more than is required to typeset any sheet music. Second, after creating and displaying the PDF file, the user could select note(s), bring up a GUI of note properties, then alter note parameters (e.g., duration, upper trill pitch). The resulting modifications would be saved to a separate file, thereby preserving the distinction between composition and performance annotations. It could be very interesting to compare performance annotations from different musicians.

Allowing users to manually tweak the interpretation of musical notation is better than nothing, but this is still a stop-gap measure. The final and most challenging step in the quest to autonomously produce good violin music is to produce a good interpretation automatically. How should we map a particular set of symbols in the musical score into low-level physical actions, without requiring any human input?

A complete answer to this problem is decades away. The main thrust of research in this field takes place at the Rencon (musical performance rendering contest) workshop, an international competition where computers attempt to perform piano music in a human-like manner (Katayose et al. 2012). The workshop has a tongue-in-cheek goal of winning the Tchaikovsky and Chopin piano contests² in

²This is likely inspired by the RoboCup robotics competition, whose stated goal is for a team of autonomous humanoid robots to win a football match against the most recent (human) World Cup football winners.

2052. At the present time, the top entries to Rencon perform piano similarly to a piano student with 2–4 years of experience. However, the range of actions is much more limited on a piano: Each note has a fixed pitch, the note can be played at different velocities (usually discretized into 127 values), at various points in time (1ms being sufficient accuracy). The state-space of piano performances of a musical score is much smaller than the state-space of violin performances.

There are two “traditional” methods of performing expressive music with a computer. The first is to hard-code musical performance rules such as “decrease the duration of notes whose pitches are increasing” or “play long notes louder than short notes”. The exact effect these modifications must be specified, either manually (Friberg et al. 2000) or interactively tuned with a real-time sequencer (Friberg 2006). However, fixed rules such as these do not match the range of expressive alterations which musicians perform; systems using fixed rules receive low score in the Rencon contest. The second method is to train a computer to determine performance rules itself, by extracting “expressive” events (e.g., note velocities, timing deviations) from a corpus of recorded performances by real musicians. These recorded performances may involve only audio files (in which case audio analysis tools are required), or they may begin from a MIDI recording of music (using existing hardware tools for recording keyboard MIDI events). These recorded performances and the original score form the training dataset for machine learning; once trained, the model predicts note alterations for new musical scores. One example of this method was used to perform Chopin music (Widmer et al. 2009).

The extraction of violinist control data from raw audio is a difficult task (Pérez & Wanderley 2012). Rather than relying on such uncertain data, I propose to again take inspiration from real-world pedagogy: *Vivi* could be trained interactively with a human teacher. In this case, the teacher would specify alterations for each note (e.g., “play these notes shorter, play this note louder, speed up over this range of notes”). Machine learning would then be used to link the musical score with the teacher’s stylistic instructions. Ideally, the machine learning would then learn performance rules (e.g., length of staccato notes, when to alter the tempo) from the teacher’s instructions. Such a system may require a few hours of human training to learn how to perform each piece of music, but this is still less work than a beginning human student. Provided that there was an easy-to-use GUI, this training could be performed by a music teacher. As the corpus of teacher-trained music grows, *Vivi* should become better at “anticipating” the teacher’s instructions (due to the machine learning). If the metadata for each piece of music was included (e.g., composer, date of composition), then *Vivi* may even be able to “learn” idiom-specific performance styles (e.g., Baroque style, Romantic style).

Chapter 7

Implementation of *Vivi, the Virtual Violinist*

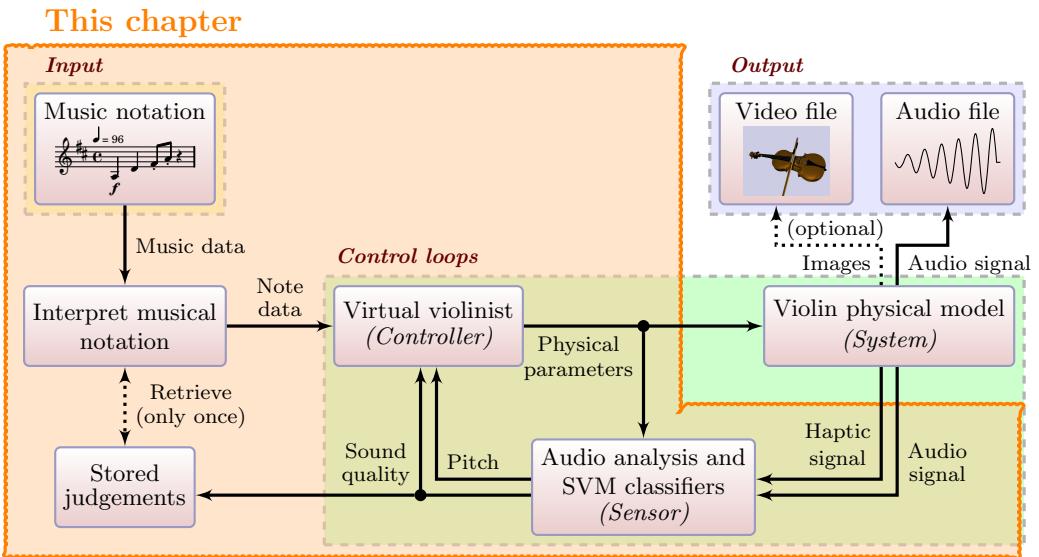


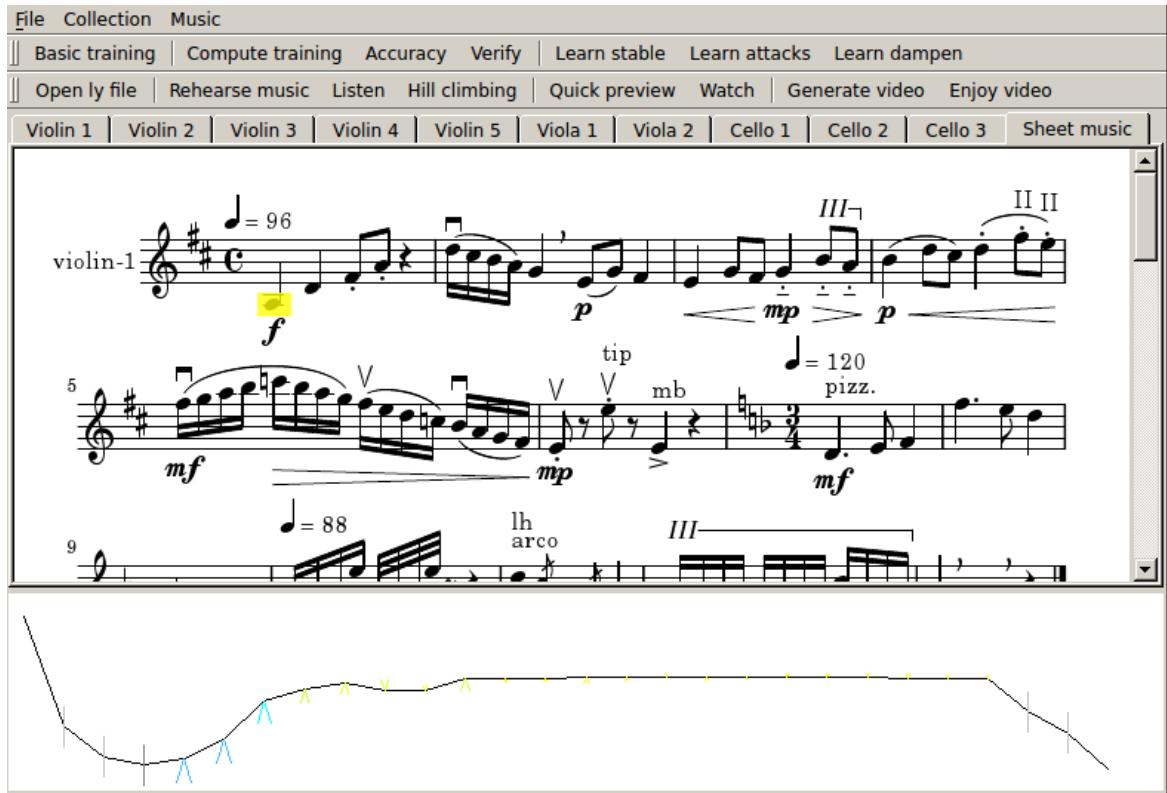
Figure 7.1: Context of this chapter.

The past two chapters described the control loop and musical interpretation. This chapter discusses all remaining issues concerning the virtual musician: Technical details of extracting information from musical scores, additional features to make *Vivi* more useful outside of a research project, and the overall design and implementation details of *Vivi*.

The *Vivi* software has two main goals: Training & calibrating the virtual violinist (as discussed in Section 6.1), and performing music (as discussed in Section 6.2). The two main graphical displays of *Vivi* are shown in Figure 7.2. These graphical displays are intended to allow users with average technical skills (i.e. an interested musician with no programming experience) to train *Vivi*. The performance of individual notes can be examined in greater detail by clicking on them in the score, and corrections to the computer's bow force judgements can be made. The software was designed to make full use of modern multi-core desktop and laptop computers. Large computations are split into multiple execution units for parallel execution.

G				D				A				E			
f	mf	mp	p												
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
2.1 N	1.5 N	1.3 N	.76 N	3.2 N	2.1 N	.82 N	1.1 N	1.8 N	.45 N	.23 N	.90 N	1.0 N	.76 N	.51 N	.41 N
2.0 N	1.6 N	.50 N	1.4 N	2.9 N	1.0 N	1.0 N	.57 N	1.2 N	.60 N	.60 N	.61 N	.77 N	.74 N	.38 N	.31 N
2.6 N	.08 N	1.3 N	.82 N	2.5 N	.84 N	.84 N	.33 N	1.9 N	.82 N	.82 N	.22 N	.87 N	.05 N	.38 N	.20 N
1.44	1.38	1.38	1.50	1.25	1.44	1.44	1.25	1.13	1.44	1.44	1.38	1.44	1.38	1.44	1.38
1.38	1.38	1.44	1.25	1.19	1.32	1.44	1.32	1.44	1.44	1.50	1.50	1.38	1.25	1.25	1.50
1.44	1.44	1.50	1.44	1.44	1.44	1.50	1.38	1.50	1.44	1.25	1.25	1.32	1.01	1.38	1.19
0.50	0.25	0.12	0.75	0.38	0.12	0.12	0.12	0.12	0.12	0.12	0.25	0.12	0.12	0.12	0.12
Mod															

(a) Main window. The tabs can select each instrument for training or the sheet music for performance. To the right of each string letter: The number of labelled audio files and the SVM cross-correlation accuracy. Below of each dynamic: Whether the dynamic passes the “verify SVMs” (Yes/No), F_b^i for the three basic finger positions, K for the three basic finger positions, and D_b .



(b) Examining sheet music. Middle: Entire score with one note selected. Below: Closer examination of the bow force and sound quality judgements for that note.

Figure 7.2: Screenshots of Vivi.

7.1 Extracting events from notation

As discussed in Section 6.2, *Vivi* uses sheet music written in the LilyPond format. This section details how information is extracted from LilyPond scores, how some of that information is used to create “clickable” PDF sheet music, and examines how music from other sheet music software can be used.

7.1.1 Extracting notation events from LilyPond

Since LilyPond input files are text, a parser could be written to directly process simple files. However, the input format allows users to write music using macros and functions as shown in Figure 7.3. In that example, the function is already built into LilyPond, but user-defined functions are also supported. Processing such files would require a much more complicated parser than a parser for files containing merely notes and rhythms.

Fortunately, LilyPond was designed to be easily extended with the Scheme programming language (a dialect of lisp). This allowed me to attach custom functions to LilyPond’s internal processing rather than attempting to parse files. LilyPond creates output (typically graphical, but also MIDI) via `Engraver` objects. These occur relatively late in the processing of a file, after any macro expansion

```
theme = \relative c' { c4\f\> d-( e-) f\p }
\new Staff {
  \tempo 4 = 72
  \theme
  \inversion c' c' \theme
}
```



(a) Text input and graphical output. The `theme` macro is defined, used in the score, then passed into the `\inversion` function which inverts the pitches in the music.

0.000	tempo	288.0			
0.000	note	72	4	0.250	point-and-click 24 16
0.000	dynamic	f			
0.000	decresc	-1			
0.250	note	74	8	0.125	point-and-click 31 16
0.250	script	staccato			
0.250	slur	-1			
0.375	note	76	8	0.125	point-and-click 37 16
0.375	script	staccato			
0.375	slur	1			
0.500	rest	4		0.250	
0.750	note	77	4	0.250	point-and-click 45 16
0.750	dynamic	p			
1.000	note	72	4	0.250	point-and-click 24 16
1.000	dynamic	f			
1.000	decresc	-1			
1.250	note	70	8	0.125	point-and-click 31 16
1.250	script	staccato			
1.250	slur	-1			
1.375	note	68	8	0.125	point-and-click 37 16
1.375	script	staccato			
1.375	slur	1			
1.500	rest	4		0.250	
1.750	note	67	4	0.250	point-and-click 45 16
1.750	dynamic	p			

(b) Extracted `.notes` output. The first column is the time (in quarter notes) of the event. The second column is the type of event, while the remaining columns are additional information. For notes, the third column is the MIDI pitch value, the fourth and fifth columns are the duration, while the sixth column is `point-and-click` data for clickable PDFs.

Figure 7.3: LilyPond input, graphical output, and extracted music events.

or music function calls. To extract the information needed by *Vivi*, I created a new `Engraver` in LilyPond which includes `listeners` for the relevant types of music events. Whenever a particular musical event (e.g., a note, a tie, an articulation marking) is processed, one of my functions is called, which outputs information about that event to a text file. The most important excerpts of this process are shown in Figure 7.4.

```

#(define (format-moment moment)
  (exact->inexact
   (/ (ly:moment-main-numerator moment) (ly:moment-main-denominator moment)))))

#(define (make-output-string-line context values)
  "Constructs a tab-separated string beginning with the score time (derived from
  the context) and then adding all the values. The string ends with a newline."
  (let* ((moment (ly:context-current-moment context)))
    (string-append
     (string-join
      (append
       (list (moment-grace->string moment))
       (map (lambda (x) (ly:format "~a" x)) values)) "\t") "\n")))

#(define (print-line context . values)
  "Prints the list of values (plus the score time) to a file, and optionally outputs
  to the console as well. context may be specified as an engraver for convenience."
  (if (ly:translator? context)
      (set! context (ly:translator-context context)))
  (let* ((p (open-file (filename-from-staffname context) "a")))
    ;; for regtest comparison
    (if (defined? 'EVENT_LISTENER_CONSOLE_OUTPUT)
        (ly:progress (make-output-string-line context values)))
    (display (make-output-string-line context values) p)
    (close p)))

#(define (format-rest engraver event)
  (print-line engraver
             "rest"
             (ly:duration->string (ly:event-property event 'duration))
             (format-moment (ly:duration-length
                             (ly:event-property event 'duration))))))

#(define (format-note engraver event)
  (let* ((origin (ly:input-file-line-char-column
                  (ly:event-property event 'origin))))
    (print-line engraver
               "note"
               ;; get a MIDI pitch value.
               (+ 60 (ly:pitch-semitones (ly:event-property event 'pitch)))
               (ly:duration->string (ly:event-property event 'duration))
               (format-moment (ly:duration-length
                               (ly:event-property event 'duration)))
               ;; point and click info
               (ly:format "point-and-click ~a ~a" (caddr origin) (cadr origin)))))

\layout { \context { \Voice
  \consists #(make-engraver
              (listeners
               (rest-event . format-rest)
               (note-event . format-note)
               ...
              )
  )
}

```

Figure 7.4: Excerpts of `event-listener.ly`. To allow other researchers to extract musical events, I added this code to the main LilyPond source code in version 2.15.0.

7.1.2 Clickable scores

As shown in Figure 7.2, the human teacher can click on any note in a score to view the bow force and sound quality judgements. This is achieved through the “point and click” functionality of LilyPond: Every graphical element in the PDF output file has a link to the position (line and column) in the input file which created that output, such as

```
textedit:///home/gperciva/tmp/example.ly:1:23:23
```

This functionality is intended to facilitate correcting errors in the text files — if there is an error in the sheet music, the user can click on that note to load the input file in her text editor, which automatically positions the cursor at the relevant text. However, I found this feature to be very useful for *Vivi*. When *Vivi* performs a piece of sheet music, an `.actions` file (as discussed in Figure 4.2) and a `.cats` file (containing a list of the sound quality judgements) are created. In addition to the physical actions or judgements, these files contain a commented-out header at the beginning of each note; this header contains the relevant `point-and-click` information. When my custom PDF browser registers such a link, it processes the `textedit://` link by searching for that string in the headers of the `.actions` and `.cats` files, then displays the results for the user to allow interactive training as described in Section 6.1.1.

7.1.3 Extracting notation from other software

MusicXML (Good 2001) has become the de facto standard for interchange of Western sheet music between score editing programs. The format is available free of charge, and can be parsed with normal XML tools. However, direct support for MusicXML is not ideal for *Vivi*.

Translating from MusicXML to *Vivi*'s `.notes` format would not be challenging, but a PDF generated with a different music engraving program would likely lack the `textedit://` point-and-click data — I am not aware of any non-LilyPond sheet music PDF generator which adds links to note-heads. Such links could be added manually in various PDF editors, but that would be an awkward and time-consuming process. Sheet music PDFs without point-and-click data would mean that *Vivi*'s human teacher would not be able to select notes by clicking on them; an alternate method would be necessary, such as using the arrow keys to move between notes or manually typing in a note's bar number and position within that bar. Such methods are more cumbersome than directly selecting a note.

I therefore suggest translating MusicXML files with the `musicxml2ly` conversion script included in LilyPond. Once the music is in the form of a LilyPond input file, the sheet music can be processed as usual with *Vivi*. This method relies on `musicxml2ly` being able to convert the relevant music. It is possible that some MusicXML files may not be converted flawlessly, but there is an extensive test suite¹ showing support for a wide range of notation.

Most other formats can be converted to MusicXML and then to LilyPond input. Music in MIDI format can be converted directly to LilyPond with `midi2ly`.

¹<http://lilypond.org/doc/v2.16/input/regression/musicxml/collated-files.html>

7.2 Additional features for practical use

Although *Vivi* is primarily a research project, I added two features which were relatively small extensions yet resulted in a significantly more useful program: *Vivi* can perform sheet music with multiple instruments, and can randomize performance aspects to play at a reduced skill level. These two factors can be combined to produce a very effective chorusing effect.

7.2.1 Scores with multiple instruments

Although there is a significant amount solo music written for violin, viola, or cello, there is a great deal more music written for combinations of those instruments. Bowed string instruments are often found in duets, quartets, and even octets with four violins, two violas, and two cellos. As a result of the number of instruments measured in Chapter 3, *Vivi* can perform music for all such ensembles. Creating audio for a piece of music with multiple instruments could be performed by manually running *Vivi* multiple times, with each part of the music being written in a separate input file and mixing the audio together. However, that is a cumbersome process, so I added the functionality to allow *Vivi* to perform music directly from the combined score instead of requiring the user to create separate parts. Internally, each part is rendered separately and mixed together, but this process is performed automatically without user input. The most famous ensemble piece found in the Suzuki books is the Bach Double concerto for two violins, shown in Figure 7.5.

Figure 7.5: Bach double violin concerto with continuo.

Audio 7.1: Bach double violin concerto introduction

<http://percival-music.ca/dissertation/a.7.1.bach-double-intro.wav>

Video 7.1: Bach double violin concerto introduction, video quality 0 and 1

<http://percival-music.ca/dissertation/v.7.1.bach-double-intro-preview.avi>

<http://percival-music.ca/dissertation/v.7.1.bach-double-intro-movie.avi>

Ensemble	Panning (from -1 to 1)
Violin 1, Violin 2	-0.5, 0.5
Violin 1, Violin 2, Cello 1	-0.5, 0.5, 0
Violin 1, Violin 2, Viola 1, Cello 1	-0.6, -0.2, 0.2, 0.6
Violin 1, Violin 2, Violin 3, Violin 4, Viola 1, Cello 1	-0.6, 0.6, -0.4, 0.4, -0.2, 0.2

Table 7.1: Default panning for multi-instrument ensembles

When *Vivi* is given a musical score, the appropriate instrument is selected based on the staff name. This applies to scores with a single instrument (e.g., solo cello) in addition to group music. The .notes for each instrument are separated and processed independently. Once all parts have been generated, they are mixed together, with the mono audio parts being panned as shown in Table 7.1. For combinations of instruments not shown in Table 7.1, the panning of each part is determined by linearly interpolating from -1 to 1 by the number of parts. If the ensemble is not listed in that table, then the panning of each instrument is linearly interpolated from left to right, in order of violins, violas, and cellos, with instrument part numbers increasing from left to right.

If the score contains cellos and any other instruments, the cello parts are amplified by -4.4 dB to prevent the cello sound from dominating the violin and viola sound. This matches real musicians; beginning cellists are often told that they cannot play “solo volume” in ensembles and must develop a “quartet volume” or “orchestra volume”.

Video is produced in a similar manner to audio: Individual images are generated for each instrument, then the separate images are concatenated. For two instruments, the images are concatenated left and right; three instruments are concatenated with the top half of the screen being instruments one and two, with the third instrument centred on the bottom; four instruments are concatenated in a square pattern. Video is not generated for more than four instruments, as such attempts showed that the resulting mixture of images was too confusing.

7.2.2 Reducing skill level

Vivi sounds like a beginning student, not a professional musician. However, the intonation is much better than a normal student, and the timing is completely metronomic. I sometimes found the discrepancy between rhythmic perfection and average tone quality to be troubling. Improving the tone

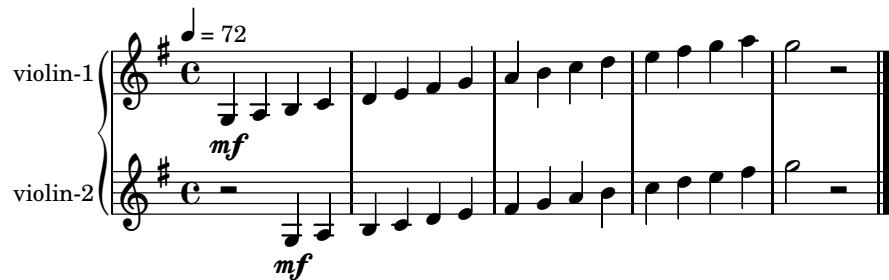


Figure 7.6: Scale in thirds with two violins. This is a traditional exercise for a violin student and teacher.

Audio 7.2: Three different skill levels of *Vivi*

<http://percival-music.ca/dissertation/a.7.2.scale-double-skill-0.wav>
<http://percival-music.ca/dissertation/a.7.2.scale-double-skill-2.wav>
<http://percival-music.ca/dissertation/a.7.2.scale-double-skill-4.wav>

quality is a challenging task, so it may therefore be desirable to deliberately degrade the intonation and timing accuracy so that the final audio sounds more consistently like a beginning student. Examples are given in Figure 7.6.

The user may select a skill level from 0 to 4; 0 being a perfect student and 4 being an extremely poor student. Selecting a skill level above 0 disables the pitch feedback loop. The reference midi M_r used to calculate the initial finger position and onset time o_t (in seconds) of each note is set to:

$$M_r = \bar{\mathcal{N}}(M_{r_0}, 0.05 \cdot s_k) \quad (7.1)$$

$$o_t = \bar{\mathcal{N}}(o_{t_0}, 0.01 \cdot s_k) \quad (7.2)$$

where $\bar{\mathcal{N}}(\mu, \sigma)$ is a “bounded” Normal distribution with mean μ and standard deviation σ , in which any result which is more than 3σ away from the mean is rejected and a new random value is chosen. The constants 0.05 and 0.1 were chosen experimentally as producing beginning violin music of the expected (lack of) skill.

7.2.3 Chorusing effect

In addition to imitating beginning music students, the deviations in pitch and timing can be used to produce a chorus effect. This phenomenon arises when listening to a number of sound sources with similar timbre, pitches, and timing — most notably in a vocal choir or a string section of an orchestra. The sounds are perceived to “blend together”, giving the impression of listening to a single musical line. The timbre of the “blended” sound is noticeably different from the solo sounds. For example, many people find the music produced by an amateur choir is often more pleasing than the music produced by a single amateur singer.

Vivi can take advantage of this technique by performing sheet music with multiple parts and slightly reducing the skill of each part. Due to the physical modelling and measurements of physical constants from multiple instruments, each part has a distinct timbre. An example is shown in Figure 7.7.

Figure 7.7: Scale arranged for chorus of all instruments. The dynamics are set to counter-balance the number of instruments (five violins vs. two violas vs. three cellos). There is a slight deviation between this figure and the actual sheet music used to generate the “chorus” audio: The score given to *Vivi* contains ten instrument staves, not three as shown here.

Audio 7.3: Scale arranged as a trio; performed by solo instruments vs a chorus of ten instruments.

<http://percival-music.ca/dissertation/a.7.3.scale-trio-skill-0.wav>

<http://percival-music.ca/dissertation/a.7.3.scale-chorus-skill-0.wav>

<http://percival-music.ca/dissertation/a.7.3.scale-chorus-skill-1.wav>

7.3 Design of *Vivi*

This section discusses the software design and implementation of *Vivi*. Design considerations include the choice of programming languages, software libraries, and effective use of modern multi-core computers. Benchmarks were performed to check the speed of the implementation. Finally, the use of *Vivi* to control realtime input is discussed.

7.3.1 Languages and libraries used

I used a clear division of programming languages for *Vivi*: The control loops were implemented in C++, while the training and musical interpretations were written in python and scheme. In terms of Figure 7.1, the “music notation” activities are performed in scheme, the “interpret musical notation” and “stored judgements” are performed in Python, and all the rest is performed in C++. The choice of each language was clear. The control loops comprise the bulk of the calculation, so they benefitted most from the speed of C++. LilyPond allows scheme extensions, so there was no alternative choice short of writing a complete input file parser. The training and musical style interpretation do not require critical speed or have any external constraints, so the ease of writing high-level code in python won out. As with the C++ implementation of Artifastring, I compiled the C++ control loop with both `g++` and `clang++` for additional warning checks, and tested all memory use with `valgrind`.

Following the suggested best practices for scientific software of “Don’t repeat yourself (or others)” (Aruliah et al. 2012, p. 3), I used open-source software libraries where possible.

Marsyas: <http://marsyas.info>

An audio feature extraction and machine learning library; used for the bowing control loop.

aubio: <http://aubio.org>

Audio feature extraction; used for the YINFFT pitch extraction in the finger control loop.

LilyPond: <http://lilypond.org>

A music engraving system; used to generate sheet music and extract musical events.

Qt4, PyQt: <http://qt-project.org> <http://www.riverbankcomputing.com/software/pyqt>

A cross-platform framework for applications; used for the GUI and easy python integration.

Poppler Qt4, python-poppler-qt4: <http://people.freedesktop.org/~aacid/docs/qt4/> <https://code.google.com/p/python-poppler-qt4/>

PDF render for Qt4 with python bindings; used to display the sheet music.

SciPy: <http://scipy.org>

Scientific tools for python; used for various statistical tools in the calibration and hill climbing.

7.3.2 Parallel processing: Queue, dispatcher, workers, and multi-stage jobs

The Artifastring library made use of parallelism in terms of CPU registers: SIMD instructions perform operations on four single-precision floats in parallel (details in Section 4.2.2). However, there is another form of parallelism: Splitting computations over multiple CPUs or possibly even over multiple computers. Theoretically, if no information needs to be shared between tasks, splitting the work between N cores can result in a factor of N reduction in time. In practice, there is always some overhead, but this is still a useful tool.

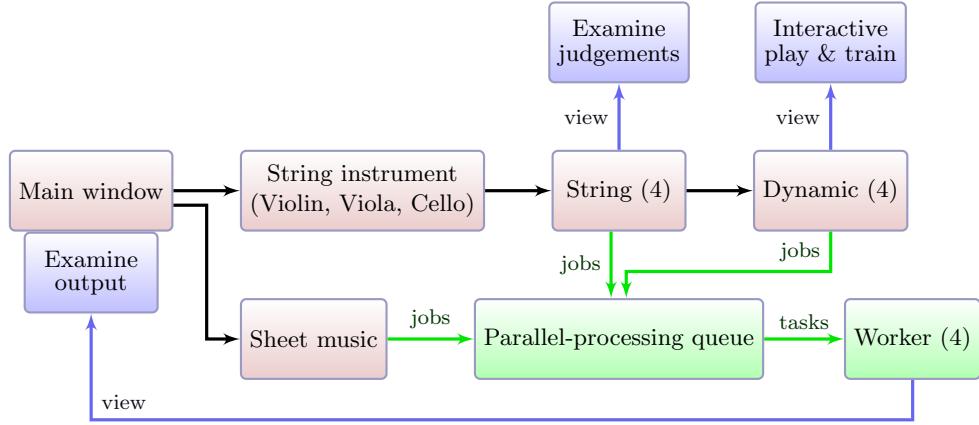


Figure 7.8: Design diagram of *Vivi*. Pink boxes represent the main objects which store data. Green boxes represent the parallel processing objects which involve significant calculations. Blue boxes represent visual output or interactive activities which depend on human input.

In the case of the training and calibration discussed in Section 6.1 and performing music with multiple instruments discussed in Section 7.2.1, the bulk of the processing does not require any information from other tasks. When simulating hundreds of audio files to perform a grid search, no audio file depends on other audio files. Such tasks are described as “embarrassingly parallel”.

The overall design of *Vivi* is shown in Figure 7.8. Activities which depend on human input are in blue, and activities which depend on significant CPU computations are in green. I refer to the latter activities as “jobs”. Any number of jobs may be triggered in the GUI; the jobs are sent to a FIFO queue and distributed between the workers. The number of workers is set to the number of processing units available to the user. On most current desktop and laptop computers, there are four processing units available (two CPU cores, each with hyper-threading). The number of workers in *Vivi* can be changed to accommodate systems with more or fewer processing units, or otherwise reduced to avoid *Vivi* using system resources desired for other purposes. Once a worker has finished one task, a dispatcher gives it another task from the queue.

There are three types of jobs which are fed to the parallel processing:

String jobs: Train SVM, compute SVM accuracy.

Dynamic jobs: Verify SVM output, calibrate F_b^i and K , calibrate D_b .

Score jobs: Generate score, render audio part x , mix audio parts, play audio, render video part x , mix video, play video, practice audio.

Most jobs are sent to a single worker, but a few jobs are split between multiple workers. I refer to one unit of such processing as a “task”. The jobs which are split into multiple tasks are: Calibrating F_b^i and K (three tasks, one for each “basic” finger position), rendering audio for scores with multiple instruments (one task per part), rendering video for each part (four tasks per part, splitting the number of frames equally between each task), and hill climbing (one hundred tasks). The video generation and hill climbing are split into many more tasks than other jobs because they require a great deal more computation. Rendering a single part of audio takes a few seconds; if a processing unit lies idle for this time due to coarse granularity of task division, there is no great loss. However, generating high-quality video can take hours; failing to utilize a processing unit will greatly increase the time required.

A few jobs require two stages: Mixing audio parts, mixing video parts, and performing the hill climbing. Using the MapReduce technique, the main task of all these jobs generates the bulk of the data, then a second task automatically combines the results of the first task.

There is one final technical detail of the parallel processing: Python includes a Global Interpreter Lock (GIL), meaning that only one python thread can execute at the same time (Beazley 2010). This decision greatly simplified many low-level implementation details while adding multi-threaded capability to the python interpreter in 1992, but it poses a problem for parallel computation. The GIL can be avoided by using separate processes, rather than separate threads. The use of distinct processes means that information cannot be shared as easily as between distinct threads, but as my design relies on queues, it poses no problem for *Vivi*.

7.3.3 Benchmarks and profiling

Benchmarks were performed using the music in Figure 7.9 for violin and the same piece transposed down a twelfth for cello. There was no measurable difference in speed between the `g++` or `clang++` compilers, since the vast majority of the processing time for running *Vivi* occurs in externally-compiled code (i.e. Artifastring, FFTW, Marsyas, and the Python interpreter). Recompiling the external code with different compilers would probably produce a difference in speed, but such testing lies outside of the scope of this dissertation, with the exception of Artifastring which was discussed in Section 4.3.2.

Results are shown in Table 7.2. The relative speed between CPUs match the earlier benchmark of Artifastring-only testing in Table 4.2. While the Artifastring test exceeded 100 times realtime for the cello simulation, *Vivi*'s fastest processing is 50 times realtime. The overall speed will depend a great deal on the number of *pizzicato* notes, rests, and most importantly, string crossings. The first half of Suzuki book 1 music only uses two strings; simulating that music will remove half of the string simulation calculations as would be required for music using all four strings due to the “turning off the string” explained in Section 2.1.4. The amount of processing required for the instrument body does not change based on the number of vibrating strings.

Figure 7.9: Black-box file for testing, benchmarks, and profiling. The simulated audio is 28.2 seconds. Audio 7.4: Black-box testing file, violin and cello.

<http://percival-music.ca/dissertation/a.7.4.black-box-violin.wav>

<http://percival-music.ca/dissertation/a.7.4.black-box-cello.wav>

CPU	Year	Clock	OS	time (seconds)			
				<i>Vivi</i>		Modelling only	
				Violin	Cello	Violin	Cello
Intel(R) Core(TM) i5-2415M	2011	2.30 GHz	64-bit	0.67	0.53	0.39	0.28
Intel(R) Core(TM)2 Quad Q9550	2009	2.83 GHz	32-bit	1.0	0.85	0.61	0.47
Intel(R) Atom(TM) N570	2011	1.66 GHz	64-bit	4.2	3.2	2.5	1.7

Table 7.2: Benchmarks of *Vivi* on common CPUs. All computers ran Ubuntu 12.04.1. CPU frequency scaling was set to maximum performance. Each test was performed twice to ensure that the executable was cached, then the mean of the next three times was recorded. The compiler version was the default for Ubuntu 12.04.1, namely g++ 4.6.3. The columns labelled “modelling only” were timed by running Artifastring’s `actions2wav` on the `.actions` file written as a by-product of *Vivi*. This allows us to estimate the amount of time spent on the feedback control rather than the physical modelling.

Method or library	Time spent	Method or library	Time spent
<code>AS::tick_bow()</code>	41.74%	<code>libstdc++</code>	3.56%
<code>libfftw3f</code>	16.89%	<code>libaubio</code>	2.05%
<code>libmarsyas</code>	12.42%	<code>AS::fill_buffer()</code>	0.99%
<code>libm</code>	7.55%	<code>AI::handle_buffer()</code>	0.64%
<code>AC::process()</code>	7.31%	<code>AS::update_bow_accel()</code>	0.47%
<code>libc</code>	5.91%	<code>Ears::listen()</code>	0.37%
total	91.71%	total	8.08%

Table 7.3: Profiling *Vivi* to show the bottlenecks. The slowest 12 functions and libraries are shown, comprising 99.79% of the processing time. Benchmarking performed on the i5-2415M CPU with the additional compiler option `-g` to display detailed information about the time spent on each line of code. Abbreviations used for objects: AS is `ArtifastringString`, AC is `ArtifastringConvolution`, Ears is the “glue” code between Artifastring and Marsyas. Artifastring is not the only library using `libfftw3f`; the Aubio pitch estimation library also relies on this library. A more detailed breakdown of the time spent on each function (not included here) showed that 67.55% of the total time was spent on Artifastring (including the relevant part of `libfftw3f` processing).

Due to the mixture of programming languages and the desire to profile the “musical performance” rather than the program start-up time, it was not feasible to measure the black-box test directly. Instead, the profiling was performed on a simpler, hard-coded C++ test consisting of 120 notes of duration 0.5 seconds each, with 30 notes on each string. Table 7.3 shows detailed profiling of time spent with `valgrind --tool=callgrind`.

As discussed in Section 4.3, the Artifastring library is extremely fast and cannot easily be further optimized. This suggests that *Vivi* could afford to use a more computationally intensive form of machine learning. There is no a priori guideline on how much computation should be spent on the virtual instrument rather than the virtual musician, but I would not query a system where the musician required ten times as much computation as the physical model.

7.3.4 Realtime interactive use of *Vivi*

Although the main use of *Vivi* is to perform musical scores, it can be instructive to experiment with aspects of the machine learning with an interactive program. *Vivi* can be controlled in realtime with an interactive python script or a tablet computer like Artifastring (as discussed in Section 4.3.4).

The `vivi_interactive.py` script allows the user to alter the string, finger position, bow position, velocity, or force. The control loops continually adjust the finger position and force. When the user specifies a finger position or force, the new value takes effect immediately, but is subject to automatic

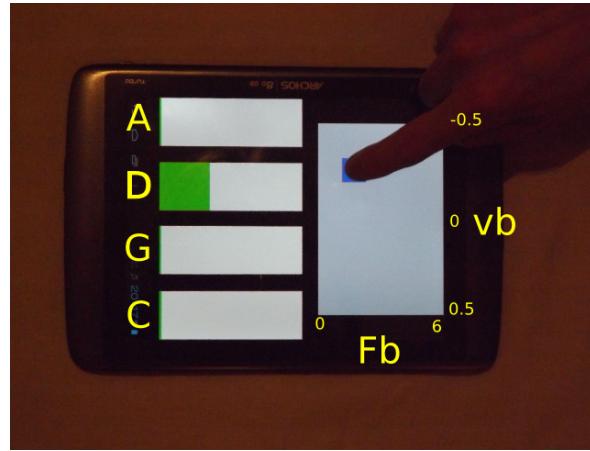


Figure 7.10: Interactive use of force correction with a tablet, violin and cello. The interface is the same as the realtime control of Artifastring in Figure 4.9, but the yellow text in this case refers to the controls for a cello. The two control loops automatically adjust the finger position x_g and bow force F_b .

Video 7.2: Interactive control of *Vivi* with a tablet

<http://percival-music.ca/dissertation/v.7.2.vivi-interactive.avi>

adjustment by the control loops in subsequent ticks. The reference pitch for the finger is assumed to be the nearest MIDI pitch value; this can be disabled if so desired.

An example of interactive use with a tablet is shown in Figure 7.10. When the user sets F_b too high, there is a “crunchy” sound for 50-100 ms, but usually the control loop quickly reduces F_b , producing a note with acceptable tone. This results in the listener musically perceiving an accented note (albeit a rather ugly accent), rather than the constant crackling noise which is produced when controlling Artifastring without the bow control loop. Similar adjustment occur if F_b is too low; there is a “wisp” beginning to a note but usually F_b is quickly increased as appropriate.

The bow control in Video 7.2 is not perfect due to the combination of dynamic, bow velocity, and timbre. The dynamic is set to f , but in Figure 7.10 the bow velocity is under the control of the user. Recall that the normal preset velocity for f is ± 0.4 m/s, and that v_b is part of the feature vectors used in the machine learning. The y axis of the bow control area varies from -0.5 m/s to 0.5 m/s; if the bow control focus is not near the edges of the area, v_b is likely too small for the expected f dynamic. One improvement would be to map the y axis to dynamic (which then implicitly sets x_b and v_b) rather than retaining the direct control over v_b as was done in the control of Artifastring in Figure 4.9.

The finger control also has some problems: When suddenly changing from one string to another, the previous string is still vibrating and the pitch detection sometimes registers that pitch. For example, 15 seconds into Video 7.2, the string is changed from G to E. The pitch detector still recognizes the much louder G string, and rapidly increases the finger position on the E string causing a squeaking sound. This is not a significant problem during performance of sheet music due to the D_b bow lifting factor. At 30 seconds, the pitch control is working well: Repeated notes with small adjustments of x_g produce either the same pitch or the upper (or lower) MIDI pitch, depending on the rounding.

There are two methods of fixing the pitch-correction problem: First, as suggested in Section 5.4.3, the physical model could provide individual $A_s[t]$ and $H_s[t]$ signals for each string s . This would break the “realism” idea of having a single audio output for the instrument, but it would completely solve

this problem. Second, a source separation algorithm could be used to estimate the audio output of each string given the combined output. Source separation is an active area of research and is not likely to produce completely accurate output, but this would maintain the “realism” of a single audio output from the physical model.

7.4 Final remarks on *Vivi, the Virtual Violinist*

This chapter described the extraction of information from the LilyPond sheet music engraver, detailed the performance of scores with multiple instruments, and examined the implementation of *Vivi* in C++ and python.

The main research contributions of this chapter are:

- Software infrastructure to present musicians with a “clickable” score which facilitates easy re-training of incorrect timbres.
- A method of reducing the apparent skill level of performances by adding random offsets to pitch and note onsets, thereby making the audio seem more “human”.

The decision to use of LilyPond rather than MusicXML directly was based in large part on my familiarity with LilyPond. It would be possible to implement such functionality in other open-source sheet music programs, such as Rosegarden or MuseScore. Alternatively, it may be possible to create “clickable” scores with commercial programs in some manner of which I am not yet aware.

The constants used for the “skill level” randomness were ad-hoc and could benefit from a great deal of research. Do student pitches and timing follow a normal distribution, and if so, what are the standard deviations for different levels of skill? Are student errors randomly distributed throughout the entire sheet music (unlikely), or are certain errors more common? A detailed study of performances by music students of various skill levels could reveal a very useful set of “human-like performance” rules which could be applied to *Vivi*’s output.

There were a number of design decisions in the implementation of *Vivi*, but these are fairly straightforward programming decisions, rather than research questions. The features are not complex, and the computation requirements for *Vivi* (as opposed to Artifastring) are insignificant. I chose to use python / scipy due to my familiarity with the language, but there was no compelling advantage of python which would outweigh another programmer’s own language of choice.

This section compares the final system with alternate methods of generating audio from sheet music and discusses potential future work.

7.4.1 Comparison of *Vivi* with alternate performance methods

Having created the virtual instruments and virtual musician, I compare the system with the obvious alternatives: Computer MIDI synthesis and human musicians. I also examine the sound quality of the physical modelling.



Figure 7.11: Comparison of *Vivi* and MIDI. Mozart’s Eine kleine Nachtmusik is shown here; chords and double stops were removed in this arrangement. The MIDI was rendered with `articulate.ly` from LilyPond 2.16.0 and the default MIDI-handling packages from Ubuntu 12.04.1: TiMidity++ 2.13.2 and the fluidr3_gm 3.1 soundfont. Reverb was disabled by calling `timidity -Ow2 -Efreverb=0 filename.midi`. Other musical excerpts are the Bach Double Violin concerto from Figure 7.5 and the scale arranged for string chorus from Figure 7.7.

Audio 7.5: Scale arranged as a chorus, MIDI and *Vivi* performances

<http://percival-music.ca/dissertation/a.7.5.scale-chorus-midi.wav>
<http://percival-music.ca/dissertation/a.7.5.scale-chorus-skill-0.wav>

Audio 7.6: Bach’s Double Violin concerto, MIDI and *Vivi* performances

<http://percival-music.ca/dissertation/a.7.6.bach-double-intro-midi.wav>
<http://percival-music.ca/dissertation/a.7.6.bach-double-intro.wav>

Audio 7.7: Mozart’s Eine kleine Nachtmusik, MIDI and *Vivi* performances

<http://percival-music.ca/dissertation/a.7.7.eine-kleine-nachtmusik-intro-midi.wav>
<http://percival-music.ca/dissertation/a.7.7.eine-kleine-nachtmusik-intro-vivi.wav>

MIDI

Sheet music engraving programs use MIDI with sampling synthesis as the default audio output. As mentioned in Section 1.2.3, this method works well for discrete-excitation instruments (e.g., piano, drums, guitar), but is poor for continuous-excitation instruments such as the violin. Comparisons of MIDI and *Vivi* are given in Figure 7.11.

Audio 7.5 contrasts the two performance methods with a scale arranged for chorus. The samples used for the MIDI rendering include vibrato, which is appropriate for many cases of string performance but not often used in scales as vibrato can mask intonation mistakes. In addition, all of the notes in

the MIDI rendition have the same timbre. Each note has the same timbre throughout the sustained portion of the note, and for each instrument, all notes sound the same with the exception of their pitches. By contrast, each note of *Vivi*'s performance is different. Some of these differences are not desirable (for example, when a note is accented more than another, or when a note fails to “speak”), but even the mistakes serve to make the performance more human-like. These variations in timbre are not appropriate for all applications — for example, the musical genre of electronica deliberately embraces artificial sounds. But since classical music was composed with variable-timbre instruments in mind, the small variations in timbre of *Vivi* are quite appropriate.

Audio 7.6 reveals another problem with MIDI rendering: Perceptual onset time. In the default sample library in Ubuntu, each note has a gradual attack over 50–70 ms, with vibrato widening over the course of this attack. With slow notes, this slow attack is not a problem, but fast notes can appear to be played late. This is the case in the second half of the first bar of the Bach Double Violin concerto: The cello’s 16th notes appear to lag behind the second violin’s 8th notes. A different sample library may or may not exhibit the same problem. Professional audio producers rendering audio with sampling synthesis will spend thousands of pounds (GBP) on sample libraries, and manually choose different sound fonts for different parts of the musical score. However, my goal is to compare the automatic rendering of sheet music, so I did not manually tweak the MIDI rendering. On the other hand, *Vivi*’s rendition has a noticeably smaller frequency range. Compared to the MIDI, *Vivi* sounds as though the instruments are muted. A more thorough examination of faults with the audio from the physical modelling is performed later in this section, during the comparison between *Vivi* and human musicians.

The opening chords of Audio 7.7 sounds much better in MIDI than in *Vivi*; again revealing the reduced frequency range of the physical modelling. In addition, the first violin can be heard clearly throughout the music. However, the 16th notes seem to merge together in the MIDI and it is difficult to hear the interesting material in the lower three voices (emphasized with *crescendo*, *decrescendo*). By contrast, *Vivi*’s 16th notes have a realistic “bouncing” sound to them, and the melodic portions of the lower three voice can be heard. The first violin in *Vivi*’s performance is not always clear and is overpowered by lower instruments playing *mp* despite being marked *f*.

In summary, *Vivi* is not ready to completely replace MIDI in the composer’s toolkit, but in some cases it demonstrates a clear advantage.

Beginning viola student

By fortunate coincidence, I began teaching a new viola student as I was finishing my research. Student S was in her early 60s and had never previously played a string instrument, although she had played piano and sung in amateur choirs since early childhood. I recorded her playing after three weeks of practice; at that point, she had received 2 hours of lessons and had practiced by herself for 4 hours². I also spent 2 hours training *Vivi* to play viola. I compare *Vivi*, student S, and the human teacher (myself) in Figure 7.12.

Vivi’s performance of “Twinkle” on viola suffers from the same problems as noted earlier. The note attacks are very strong; it sounds as though every note is accented. Despite the overly high

²The duration of solo practice may seem a bit low, but I asked her to practice for no more than 10 minute at a time. Playing a stringed instrument requires a surprising amount of physical strength and agility in the left-hand fingers. Children’s bodies adapt and heal very quickly, but older adult students sometimes have difficulty developing these abilities, particularly if they already suffer from occasional arthritis.



Figure 7.12: Comparison of *Vivi* and human musicians playing “Twinkle, twinkle, little star”. The humans and *Vivi* are both playing viola-II, although the instrument received a new D string after the physical measurements were taken.

Audio 7.8: Comparison of *Vivi*, a human student, and a human teacher

<http://percival-music.ca/dissertation/a.7.8.twinkle-vivi-skill-0.wav>
<http://percival-music.ca/dissertation/a.7.8.twinkle-student-week-03.wav>
<http://percival-music.ca/dissertation/a.7.8.twinkle-teacher.wav>

initial bow forces, certain notes do not “speak” clearly, such as the first B in bar 2, the first G in bar 7, the A in bar 10, and (most unfortunately) all the notes in the last bar.

Student S adds a few pauses to the music, particularly when changing strings. In addition, she has some problems coordinating her two hands; occasionally she will begin bowing a note before adjusting her left-hand fingers (e.g., bar 3, bar 10) or even changing to the correct string (e.g., bar 6–7). There were numerous intonation mistakes, but she corrected them quickly by sliding her finger to the correct position.

The teacher’s performance has good bow tone, and any intonation mistakes are quickly fixed.

Physical modelling sound quality

There is a clear difference in audio quality between humans and *Vivi*, despite the simulation being performed with the physical constants measured from the instrument played by the humans. This difference is examined in greater detail in Figure 7.13, contrasting the first note of the teacher’s recorded audio with *Vivi*’s audio output.

In particular, the strength of modal peaks drops tremendously after about 4 kHz. In the range of 5 kHz to 8 kHz, most simulated peaks are at least 50 dB less than the modal peaks in the recorded audio. Some of this difference may arise due to faults in the measurement of the instrument body impulse responses. If the audio recording of instrument taps from Section 3.3.2 does not have a flat frequency response, those imperfections will be reflected in the simulated sound. Such problems could arise due to electrical or acoustic noise, effects of the room’s acoustics, microphone construction, or if the taps themselves were not sufficiently short (and, equivalently, spectrally broad) to contain sufficient high-frequency energy. Examining the frequency response of the silence before the impulse shows the effect of room, electrical and acoustic noise, and microphone; this is shown in Figure 7.14. There will clearly be some bias due to imperfect recording conditions, but this does not explain the difference between frequency ranges 2–4 kHz and 5–8 kHz. To further investigate the weakness of upper frequencies in the simulated audio, I disabled the instrument body filter entirely and re-synthesized the violist’s actions; this is included in Figure 7.13. In this simulation, a few modes (approximately 6.7 kHz, 7.3 kHz, and 7.6 kHz) have equal or even higher magnitude than the recorded audio, while other modes are still less than the audio. I theorize that this discrepancy in the distribution of energy amongst modes is due to the limited number of forces. Recall that while bowing the string in the

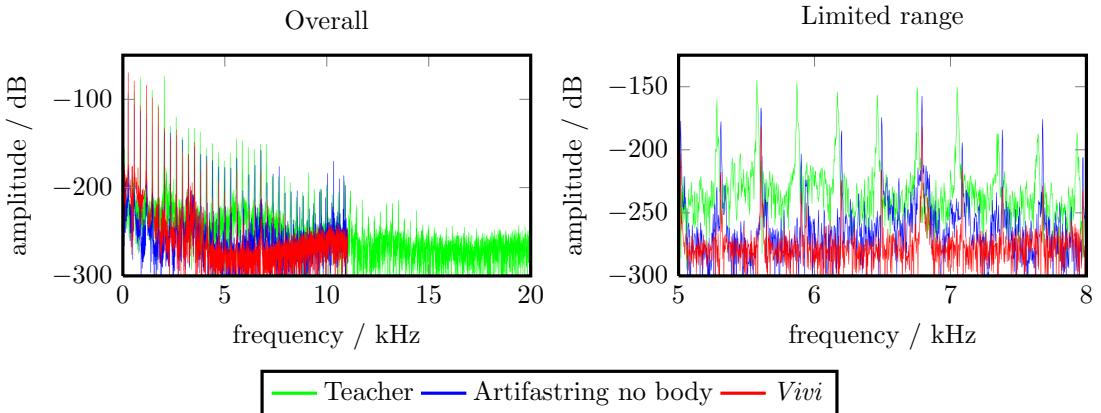


Figure 7.13: Examination of first note of “Twinkle, twinkle, little star”. The frequency difference in modes likely arises due to different string tuning; *Vivi*’s pitch (measured with YINFFT) is 293.2 ± 0.14 Hz, while the human teacher’s pitch is 292.5 ± 0.23 Hz. To investigate the difference in upper frequencies between the real and simulated instruments, the note was resynthesized in Artifastring without applying the instrument body filter. Since the Viola-D-II string is simulated at 44100 Hz, this produces some aliasing after decimating to 22050 Hz for the overall instrument output. Audio 7.9: Comparison of one note of “twinkle” between teacher and *Vivi*

```
http://percival-music.ca/dissertation/a.7.9.twinkle-teacher-note1.wav
http://percival-music.ca/dissertation/a.7.9.twinkle-vivi-note1.wav
http://percival-music.ca/dissertation/a.7.9.twinkle-artifastring-no-body-note1.wav
```

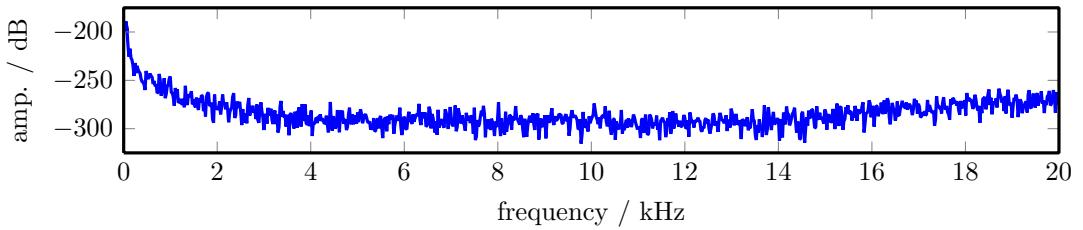


Figure 7.14: Frequency response of viola-II silence before impulse.

model, there is only one finger force and one bow force. Limiting the external forces in this manner permitted a direct solution of the bow-string friction interaction, but if the position of either point force is close to a node, the external force will have little effect on that mode. This was a compromise between physical realism and computational efficiency.

In retrospect, I suspect that I erred too much on the side of fast simulation. The extensive examination of finger forces in Section 2.2.2 and Section 2.2.3 showed that the model required at least two forces for each finger (left- and right-hand) to avoid undesirable behaviour. Some problems with the single-force bow modeling were noted in Section 2.3.1, but at the time I made the design decision to accept those problems.

7.4.2 Future work on *Vivi*

Given the previous comparisons between human performance and the *Vivi*, the ideas for improving the model in Section 2.4.1, Section 3.6.1, and Section 4.3.2 should be given more weight. However, this section will focus on the virtual musician rather than the virtual instrument.

I discussed future work on the control loops in Section 5.4.3 and improvements to the calibration and musical interpretation in Sections 6.4.1 and Section 6.4.2, so here I will restrict myself to comments about the implementation. These fall into four categories: The software design, support for multiple instruments, applying *Vivi* to large collections of sheet music, and more stringent comparisons between *Vivi* and alternate methods of generating sound.

Software design

The profile in Table 7.3 showed that there are no bottlenecks in the python code, so there is no performance reason to alter the overall design. However, it may still be desirable to shift the division between C++ and python slightly. In particular, once a musical interpretation has been created (i.e. a list of notes with their mid-level modifications has been finalized), this list could be saved to a file, which would then be processed with C++. At the moment, the musical interpretation is saved in python, and python then calls `vivi_controller::note(...)`, `vivi_controller::pizz(...)`, or `vivi_controller::rest(...)` accordingly.

Splitting the musical performance in this manner will not offer a noticeable improvement in speed, but it would result in a cleaner separation of layers. This would in turn allow easier testing, optimization, and make it simpler for other research projects to interact with *Vivi*. This interaction could take two forms: either generate their own musical interpretation and use *Vivi* for the mid- and low-level rendering, or else take the *Vivi*'s musical interpretation and render it with other means (any of the synthesis methods discussed in Section 1.2.3).

Multiple instruments

The audio performance of multi-instrument scores would be improved by simulating room acoustics instead of merely setting a left/right panning value for each part. The simulation of room acoustics is an open research problem, but a good approximation is to use first-order reflections. Given the dimensions of a room, the location of each sound source (instrument in the score), and the listener (assumed to have ears 22 cm apart), we can calculate the distance between each sound source and each listener's ear. After calculating the length of the direct path, the paths with a single reflection is calculated from the length from the sound source to a wall (or floor or ceiling) and thence to each listener's ear, assuming that the sound waves' reflection is non-diffuse.

Once the distance travelled is computed for each path from sound source to destination (direct path, plus one path including one reflection from each surface), we can calculate the time delay and amplification for each path. In particular, given the speed of sound at room temperature as $c = 340 \text{ m/s}$, the path length L_i , and sampling rate f_s , the delay d_i (in samples) and amplification a_i are

$$d_i = \frac{L_i \cdot f_s}{c} \quad a_i = \frac{1}{L_i^2} \quad (7.3)$$

The paths are summed to produce the final output. Given the sound source $x[t]$, the signal at the destination is

$$y[t] = \sum_{i=1}^N a_i \cdot x[t - d_i] \quad (7.4)$$

In the case of first-order reflections, $N = 5$ (for two-dimensional rooms) or $N = 7$ (for three-dimensional rooms). Non-integer delays can be accommodated using fractional delay lines with linear interpolation.

The animation of multi-instrument music could be improved by having multiple instrument objects in Blender, then rendering the whole scene at once instead of rendering each instrument independently. These instruments could also be placed within a larger scene, for example a virtual concert hall. For added realism, the instrument placement and dimensions of the virtual room in Blender would be used in the acoustic room simulation.

Large collections of sheet music

There are several websites with large collections of copyleft sheet music in digital format, such as MutopiaProject³ and Kern scores⁴. *Vivi* could be used to automatically produce audio performances of these works. The results would not be professional quality, but for many pieces of music it would give a better indication of the sound than a MIDI rendering would. Such an effort would benefit library users browsing for audio, as well as benefiting *Vivi* for the additional real-world testing of the music interpretation.

³<http://www.mutopiaproject.org/>

⁴<http://kern.ccarh.org/>

Chapter 8

Conclusion

I have developed a new method of synthesizing audio from sheet music. Part I of this dissertation described the creation of the virtual violin family (violin, viola, cello), while Part II discussed training a virtual musician to perform sheet music with these instruments. This conclusion will evaluate the project's goal, discuss the academic contributions of this dissertation, examine the lack of philosophical implications of virtual musicians, and end with an overview of anticipated future work.

8.1 Evaluating the initial goal

The initial goal set out in Section 1.3 was:

Given a machine-readable representation of sheet music, the computer autonomously produces audio and video that sounds as if it was performed by a human.

Sheet music can be read directly from the LilyPond format or converted from MusicXML to LilyPond, satisfying the first clause. The quality of *Vivi*'s performance is roughly on par with a music student with two years of experience. The problem definition included three constraints, which were satisfied in the following manner:

Free software: The physical modelling and virtual musician software was written by me and licensed under the GPLv3+ as discussed in Appendix C. This enshrines everybody's right to use, modify, and redistribute the code. All libraries used by Artifastring and *Vivi* are also available under copyleft licenses.

Easily extended: The different layers of the performance process — musical interpretation, mid-level note modifiers, feedback control, and physical modelling — are clearly separated with only a small amount of information travelling between layers. In addition, each step of the process is discussed in detail in this dissertation, with many graphs, diagrams, and audio examples. Interested developers can modify or replace one layer of the process without possessing any expertise or knowledge of other layers. Physical constants can be easily modified or added if more instruments are measured.

Human-like pedagogy: The only human input required is to judge the timbre in terms of bow force, as discussed in Section 5.2.1. No knowledge of programming or physics is required for training *Vivi*.

An informal comparison of simple music performed by *Vivi* and a human student was given in Section 7.4.1. Examples of entire pieces of music performed with *Vivi* are given in Appendix B. It would be interesting to conduct a formal comparison of *Vivi* and human students of varying skill levels (e.g., 3 months, 6 months, 1 year, 2 years).

Such a comparison could be designed similar to Turing’s “imitation game” (Turing 1950), often referred to as the “Turing test”. In the imitation game, a human judge converses with two unknown entities A and B via text. One of these entities is a human, while the other is a computer running a program which attempts to respond in a human-like manner. After conversing for a number of minutes, the judge must decide whether A or B is the computer. Over the years there have been many variations proposed for this game (Saygin et al. 2000).

This test could be adapted for testing *Vivi* against human performances; instead of conversing via text, the human judge would listen to an audio recording produced either via recording a human musician, or synthesized with *Vivi*. However, while envisioning such a test, there are a few different questions we could ask the judge:

- Of the audio recordings A and B, which sounds as if it was performed by a human?
- Of the audio recordings A and B, which did you enjoy listening to the most?
- Rank the musical quality of each recording from 1 to 10.

Ideally this experiment would be performed multiple times. To measure statistical significance, I suggest the Friedman test (Friedman 1937), which is a non-parametric evaluation of different treatments over multiple tests. For example, we could ask $n = 10$ human judges to listen to $k = 4$ different performances of the same piece of music; one of these performances would be from *Vivi*, while the remaining three would be from musicians of varying skill levels.

I will not delve further into the consequences of the experimental design, but will rather note that there is a fundamental difference between attempting to *imitate* a human musician and trying to automatically create *useful* audio from sheet music. With no pun intended, I suggest that the latter goal is more useful to pursue. There is certainly value in attempting to imitate human performance; to quote Nobel laureate Richard Feynman, “What I cannot create, I do not understand”¹. Writing a computer program to perform music in a particular style would be a very useful test for many theories in musicology; if the program fails to generate the expected performance, then the musical theory should be re-evaluated.

Although the advancement of musicology is a worthwhile goal, the majority of applications I envisioned in Section 1.1 are more practical. There are three main applications of virtual musicians: Automatic performance of sheet music for composers (be they professionals, students, or amateurs); automating certain aspects of instrumental control to allow non-musicians to “perform” a virtual instrument with assistive technologies; and generating audio to help users of sheet music libraries select piece(s) of music. In all these cases, the overall quality of the music is the most important factor in its applicability, not whether the musical performance are realistic or not. Realism will likely play a role in users’ opinions about the generated sound, but this can be captured by asking users to rank the *quality* of the recordings, rather than their *human-ness*.

¹Written on Feynman’s blackboard at the time of his death in 1988,
http://archives.caltech.edu/search_catalog.cfm?search_field=%2B1.10-29 accessed 2013 Jan 27.

8.2 Philosophical implications (or the lack thereof)

Regardless of whether we shift *Vivi*'s goal from "imitating a human" to "producing music that sounds better than a beginner with X weeks of experience", *Vivi* should not be subjected to the same philosophical quagmire which Turing attempted to avoid by suggesting the imitation game as a replacement for the question "can computers think". I have made no claim that *Vivi* can "feel" music, "express" music, or anything of the sort. The formal problem statement given in Section 1.3 states that the computer should produce "audio that sounds as if it was performed by a human". I am not suggesting that, if *Vivi* is capable of fulfilling that goal, we describe *Vivi* as a "musician" or "artist". Such terms are highly loaded and not at all helpful.

Rather, I claim that if *Vivi* is capable of producing audio which sounds as though it was performed by a human, then *Vivi* is capable of producing audio which sounds as though it was performed by a human. No more, no less. If there is an application scenario that would benefit from such audio (e.g., inexpensive performances of works by composition students, allowing people paralyzed from their neck down to exert some level of control over string music), then *Vivi* would be able to fulfil that need. If the application scenario requires other facets (e.g., sounding like a concert soloist, following a live human conductor), then *Vivi* would require modifications.

Debates over the definitions of words are rarely helpful, and I cannot imagine a scenario² in which *Vivi*'s output would be improved by such a debate. There are concrete flaws in certain aspects of *Vivi*, which require a combination of physics, digital signal processing, machine learning, and/or basic violin pedagogy to solve.

Leaving aside any semantic debate about terms such as "music", "musician", or "aesthetics", some people may raise an ethical question. Their argument may be something along these lines: Virtual musicians which sound as good as human musicians with 15 years of experience could replace real musicians, especially if the virtual musicians were available for free. *Vivi* is currently far from being sufficiently skilled to replace any human musicians, but this may be a concern in a decade. Is it ethically acceptable to put skilled workers out of work?

Concerns over the effect of automation are not new; the Luddite movement famously espoused these views in the early 1800s. Automation certainly reduces the amount of work required to create a product or perform a task. If the demand for the product does not increase, then fewer workers will be required.

I have two replies to these concerns. First, worries about automation in music are not new. The player piano, phonograph, and wireless radio all raised concerns about putting musicians out of work. In the present day, sampling synthesis is often used for "background" instruments in commercial music recordings. This point in itself does not mean that the concerns are meaningless, but rather this is a century-old debate to which *Vivi* does not add anything new. Second, I will note that the long-term effect of automation is overwhelmingly positive; the present quality of life in the Western world is much higher than it was two hundred years ago. We live longer, have more leisure time, have greater opportunities for travel or enjoying the arts, etc.

In summary, *Vivi* raises no new philosophical issues, and the debate about existing issues will not help *Vivi* to produce better output. My focus is creating good music, not talking about music.

²I received a first-class Honours in Philosophy, so I do not believe that I suffer from a lack of familiarity with philosophical debate.

8.3 Contributions of this dissertation

This dissertation is primarily composed of applying well-known techniques to the performance of stringed music. Although this research did not result in any fundamental discoveries in physics or machine learning, there is considerable value in dissecting an overall goal into specific problems which can then be solved with known techniques. The choice of which techniques to use, the considerations which led to those decisions, and the consequences of those decisions are all useful contributions which can shape future research on this problem.

The single largest contribution of this dissertation is the presentation of a complete system for performance of stringed music published under a permissive copyright license. This enables future researchers to test individual aspects of the process without requiring them to be experts in other areas. For example, a mechanical engineer could improve the bow-friction model, recompile Artifastring, hire a violin student to spend two hours training the new system, then listen to the Bach Double Violin concerto being performed with the new friction model. Alternatively, a computer scientist with no knowledge of physics or music could change the type of the machine learning used for the feedback control, and hear entire pieces of music being performed with that alteration within an hour. Finally, a musician with a few hours of experience learning the Python programming language — a language noted for being “readable” and easy to learn — could test different theories of instrument performance, applying the results to real string music within minutes of his modifications.

The original form of the physical modelling equations for Artifastring in Chapter 2 came from (Demoucron 2008). Those equations were extended to include damping for the left-hand finger, the addition of right-hand *pizzicato*, and a few problems in the bow-friction model were fixed, most notably avoiding an unstable system with high-Q strings. In Chapter 3, physical constants were measured from five violins, two violas, and three cellos, forming the largest collection of such constants which include modal decay values. Great care was taken to create a very efficient C++ implementation in Chapter 4, applying techniques for high-performance software such as SIMD instructions and extensive profiling. Animated videos are created automatically to enable immediate visualization of the bow and finger actions, allowing the system to serve as an example for beginner students. The final system of physical modelling can simulate ten distinct instruments, and simulating a single instrument can be performed one hundred times faster than real-time.

The work on *Vivi* began with an examination of the application of human violin pedagogy to the computer in Chapter 5. The problem of bow control was decomposed into signal feature extraction and SVM classification, while restricting the human training to that which could be reasonably performed by a musician with no physics or computer programming background. The control of the left-hand finger was achieved by PID control with a few modifications to avoid false corrections due to poor bow control. The calibration of initial parameters in Chapter 6 was solved by applying grid search using the performance metric of the previously-trained SVM classifiers. Solo practicing was implemented with the artificial technique of hill climbing, again using the SVM classifiers as the performance metric. Musical notation used in beginner string music was translated into physical actions by considering how that notation would be performed. Finally, Chapter 7 used multi-threaded design (a FIFO queue and worker threads) to minimize the running time of large operations in *Vivi* on multi-core computers.

A test suite of sheet music was constructed, consisting of solo violin music from Suzuki books 1 and 2, the first movements of the Bach Double Violin concerto and Mozart’s Eine kleine Nachtmusik for string quartet, and Pachelbel’s Canon in D.

8.4 Future work

Detailed suggestions for improvement were given at the end of each chapter. This section will not re-iterate those points, but rather will provide some thoughts about the relative benefits from working on each area.

As was discussed in the comparison of *Vivi* with a human student in Section 7.4.1, in retrospect it seems that the balance between physical realism and computation efficiency was too far on the side of efficiency. There would be two main benefits to improving the quality of the physical model: First, the audio quality would improve, but second, it is possible that certain problems with feedback control could be mitigated. Increasing the sample rate and number of modes could aid in the establishment of Helmholtz motion, and taking the bow width into account would certainly change the model’s behaviour.

Similar benefits (improved audio, improved control) could arise by a “virtual luthier” tweaking the physical constants. Real-life luthiers select materials, shapes, and modify instrument construction techniques in order to improve the sound quality. It is reasonable to assume that similar skills could also benefit virtual instruments, especially since there are no physical constraints on the materials. A virtual luthier could begin with a cello string and subject it to hundreds of Newtons of tension in order to tune it as a violin E string. Conversely, the slow modal decays of the violin E string could be applied to the cello C string without altering any other constants. Directly altering constants like this is impossible in real life, but it could result in a fascinating set of sounds from the model. Optionally, a virtual luthier could create entirely new instruments, such as an 8-stringed “octo-cello” whose strings spanned the entire range of cello and violin strings, but whose body filter was the impulse response from a double bass.

The automatic control of note attacks is the weakest part of *Vivi*; this is the main problem which brings the practical use of *Vivi* into question. Improving the control is a question of control theory, digital signal processing, and machine learning. As discussed in Section 5.4.2, we cannot expect more detailed information from the human trainers without relaxing the “human-like pedagogy” constraint, but it may be possible to use the existing machine learning to train a second layer of machine learning. This second layer would be used for the direct control of the string.

Improving the musical interpretation is a longer-term research question in artificial intelligence and musical aesthetics, and is not necessary for all applications of *Vivi*. For example, the Vocaloid computer singing software does not interpret notation; users specify the notes’ pitches, durations, and lyrics. Expressive alterations are performed manually by the “tuner” (who is occasionally a different person than the composer and producer). Similarly, in some cases of using *Vivi* as part of a virtual instrument for assistive technologies, it may be expected that the musical expression will be controlled directly by the user.

Regardless of which aspect(s) are chosen to receive future work, *Vivi* remains a platform allowing experts to focus on their specialities. Due to the permissive copyright license and available raw data in Appendix C, future researchers may directly work on their expertise without needing to re-implement areas outwith their interests.

“We can only see a short distance ahead, but we can see plenty there that needs to be done.”
 (Turing 1950, p. 460)

Appendix A

Additional Mathematics for Physical Modelling

This appendix discusses a conjecture concerning the bowing coefficient D_1 in Chapter 2, which was removed from the main text for brevity. D_1 occurs in a few bowing equations which could be simplified if $D_1 > 0$. For example, the “negative slipping” bowing case (2.41),

$$\begin{aligned}\Delta v &= \min \left(\frac{c_1 + \sqrt{c_1^2 + 4c_0 D_1}}{2D_1}, \frac{c_1 - \sqrt{c_1^2 + 4c_0 D_1}}{2D_1} \right) \quad \text{provided that } \Delta v < 0 \\ &= \frac{c_1 - \sqrt{c_1^2 + 4c_0 D_1}}{2D_1} \quad \text{provided that } \Delta v < 0 \text{ and } D_1 > 0\end{aligned}\tag{A.1}$$

While I cannot present a formal proof that $D_1 > 0$, I give strong analytic evidence and note that it is true for all of the physical constants I have measured for violin, viola, and cello strings.

Lemma 1: String coefficients $X_{3n} > 0$ and $Y_{3n} > 0$

Recall the definitions from Table 2.2, for $n \in \{1, 2, \dots, 40\}$.

$$X_{3n} = \frac{1 - \left(\cos(\omega_n dt) + \frac{r_n}{\omega_n} \sin(\omega_n dt) \right) e^{-r_n dt}}{\rho_L \omega_{0n}^2} \quad Y_{3n} = \frac{\left(\omega_n + \frac{r_n^2}{\omega_n} \right) \sin(\omega_n dt) e^{-r_n dt}}{\rho_L \omega_{0n}^2} \tag{A.2}$$

The number of modes and sampling rates were chosen such that $\omega_n dt < \pi$ (to stay below the Nyquist sampling limit). In addition, $\omega_n > 0$, $\rho_L > 0$, and $r_n \geq 0$, and so $-1 < \cos(\omega_n dt) < 1$, $0 < \sin(\omega_n dt) < 1$, and $0 < e^{-r_n dt} \leq 1$. Given those constraints, $Y_{3n} > 0$.

$X_{3n} > 0$ when

$$\left(\cos(\omega_n dt) + \frac{r_n}{\omega_n} \sin(\omega_n dt) \right) e^{-r_n dt} < 1 \tag{A.3}$$

If there were no modal decay, $r_n = 0$ and (A.3) reduces to $\cos(\omega_n dt) < 1$ which was previously noted as true. When modal decay is added with $r_n > 0$, the sine term becomes relevant, but $e^{-r_n dt}$ will decrease. I simplify (A.3) with the following substitutions

$$x = \omega_n dt \quad y = r_n dt \quad (\text{A.4})$$

which reduces the left-hand side of (A.3) and the range of x and y as

$$\left(\cos(x) + \frac{y}{x} \sin(x) \right) e^{-y} \quad \text{for } 0 < x < \pi, 0 \leq y \quad (\text{A.5})$$

Within that range, the trigonometric functions are bounded by

$$\cos(x) < 1 \quad \sin(x) < x \quad \text{for } 0 < x < \pi \quad (\text{A.6})$$

Substituting $\sin(x) = x$ and $\cos(x) = 1 - \epsilon$, where ϵ is an arbitrarily small value which is greater than zero, (A.5) can be rewritten as

$$(1 - \epsilon + y) e^{-y} \quad (\text{A.7})$$

Differentiating the result to find its maximum value,

$$\frac{d(1 - \epsilon + y)e^{-y}}{dy} = e^{-y} - (1 - \epsilon + y)e^{-y} = 0 \quad \therefore y = 0 \quad (\text{A.8})$$

This clearly gives us a maximum value of $1 - \epsilon$, satisfying the condition in (A.3) that this value be < 1 . Therefore $X_{3n} > 0$.

Lemma 2: Violinist position coefficients $A_{00}, B_{00} > 0; A_{11}, B_{11} \geq 0$

The positions x_0 and x_1 of the forces on the string are set by the actions of the violinist. These positions are used to calculate the A and B coefficients, defined in (2.20) and included here for reference:

$$A_{pq} = \sum_{n=1}^N \phi_n(x_p) \phi_n(x_q) X_{3n} \quad B_{pq} = \sum_{n=1}^N \phi_n(x_p) \phi_n(x_q) Y_{3n} \quad \text{for } p, q \in \{0, 1\} \quad (\text{A.9})$$

$$\phi_n(x) = \sqrt{\frac{2}{L}} \sin\left(n\pi \frac{x}{L}\right) \quad (\text{A.10})$$

If the finger is not on the string, $x_1 = 0$, $\phi_n(x_1) = 0$ and thus A_{11} and $B_{11} = 0$. If the finger is on the string, then $x_1 > 0$ and $\phi_n(x)$ will vary between $-\sqrt{\frac{2}{L}}$ and $\sqrt{\frac{2}{L}}$. However, if $p = q$, then $\phi_n(x_p) \phi_n(x_q) = (\phi_n(x_p))^2$, and since X_{3n} and Y_{3n} are greater than zero by lemma 1, A_{00}, A_{11}, B_{00} , and B_{11} will also be greater than zero provided that $x_p > 0$. We are only interested in D_1 when the bow is on the string, so $x_0 > 0$ and thus A_{00} and B_{00} will be greater than zero.

Lemma 3: Violinist position coefficients $A_{01} = A_{10}, B_{01} = B_{10}$

Since multiplication is commutative, $\phi_n(x_p) \phi_n(x_q) = \phi_n(x_q) \phi_n(x_p)$.

Lemma 4: $B_{00}B_{11} - B_{01}B_{10} \geq 0$

From lemma 3, we can rewrite this lemma and expand with the definition of B_{pq} ,

$$\begin{aligned} & B_{00} & B_{11} & - & (B_{01})^2 & \geq 0 \\ & \left(\sum_{n=1}^N \phi_n(x_0)\phi_n(x_0)Y_{3n} \right) & \left(\sum_{n=1}^N \phi_n(x_1)\phi_n(x_1)Y_{3n} \right) & - & \left(\sum_{n=1}^N \phi_n(x_0)\phi_n(x_1)Y_{3n} \right)^2 & \geq 0 \end{aligned} \quad (\text{A.11})$$

I define

$$a_n = \phi_n(x_0)\sqrt{Y_{3n}} \quad b_n = \phi_n(x_1)\sqrt{Y_{3n}} \quad (\text{A.12})$$

Substituting a_n and b_n into the left-hand side of (A.11) gives

$$\left(\sum_{n=1}^N a_n^2 \right) \left(\sum_{n=1}^N b_n^2 \right) - \left(\sum_{n=1}^N a_n b_n \right)^2 \geq 0 \quad (\text{A.13})$$

which is a restatement of the the Cauchy-Schwarz inequality and is thus true.

Conjecture 1: $A_{11}B_{00} - A_{10}B_{01} \geq 0$

A similar analysis to lemma 4 cannot be performed for this conjecture. In the trivial case of bowing a string without any finger, A_{11} , A_{10} , and B_{01} are all 0 and the conjecture is true. For the case of a finger on the string, I cannot present a proof but can give evidence for this conjecture.

To investigate the case of a finger on the string, we first consider a simple (unrealistic) model in which X_{3n} and Y_{3n} do not vary based on the mode but the ϕ_n values remain as stated (i.e. mode-dependent). Specifically, I replace A_{pq} and B_{pq} with \hat{A}_{pq} and \hat{B}_{pq} which use the means of X_{3n} and Y_{3n} , i.e.

$$\bar{X}_3 = \frac{1}{N} \sum_{n=1}^N X_{3n} \quad \bar{Y}_3 = \frac{1}{N} \sum_{n=1}^N Y_{3n} \quad (\text{A.14})$$

$$\hat{A}_{pq} = \sum_{n=1}^N \phi_n(x_p)\phi_n(x_q)\bar{X}_3 \quad \hat{B}_{pq} = \sum_{n=1}^N \phi_n(x_p)\phi_n(x_q)\bar{Y}_3 \quad (\text{A.15})$$

$$c_i = \phi_n(x_0) \quad d_i = \phi_n(x_1) \quad (\text{A.16})$$

We can then expand

$$\begin{aligned} \hat{A}_{11}\hat{B}_{00} - \hat{A}_{10}\hat{B}_{01} &= \left(\sum_{n=1}^N d_i d_i \bar{X}_3 \right) \cdot \left(\sum_{n=1}^N c_i c_i \bar{Y}_3 \right) - \left(\sum_{n=1}^N d_i c_i \bar{X}_3 \right) \cdot \left(\sum_{n=1}^N c_i d_i \bar{Y}_3 \right) \\ &= \bar{X}_3 \bar{Y}_3 \left[\left(\sum_{n=1}^N d_i d_i \right) \cdot \left(\sum_{n=1}^N c_i c_i \right) - \left(\sum_{n=1}^N d_i c_i \right) \cdot \left(\sum_{n=1}^N c_i d_i \right) \right] \\ &= \bar{X}_3 \bar{Y}_3 \left[\sum_{n=1}^N d_i^2 \sum_{n=1}^N c_i^2 - \left(\sum_{n=1}^N c_i d_i \right)^2 \right] \end{aligned} \quad (\text{A.17})$$

Since \bar{X}_3 and \bar{Y}_3 are both greater than 0, the Cauchy-Schwarz inequality shows that (A.17) is greater or equal to 0.

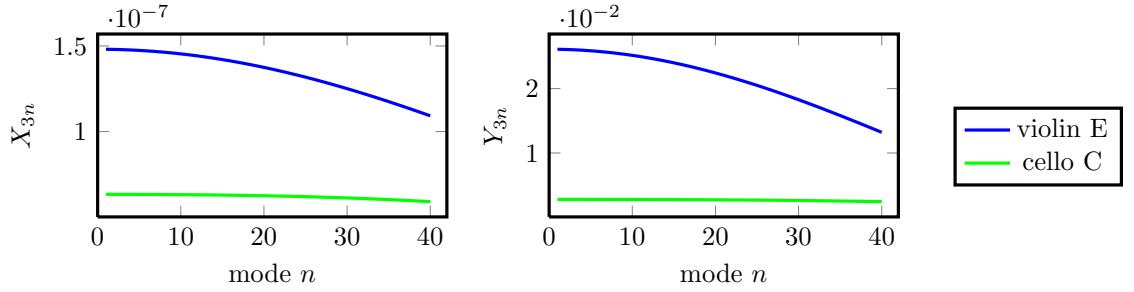


Figure A.1: X_{3n} and Y_{3n} for violin-E-I and cello-C-I. For these two instruments, $5.9 \times 10^{-8} < X_{3n} < 1.5 \times 10^{-7}$ and $2.4 \times 10^{-3} < Y_{3n} < 2.7 \times 10^{-2}$.

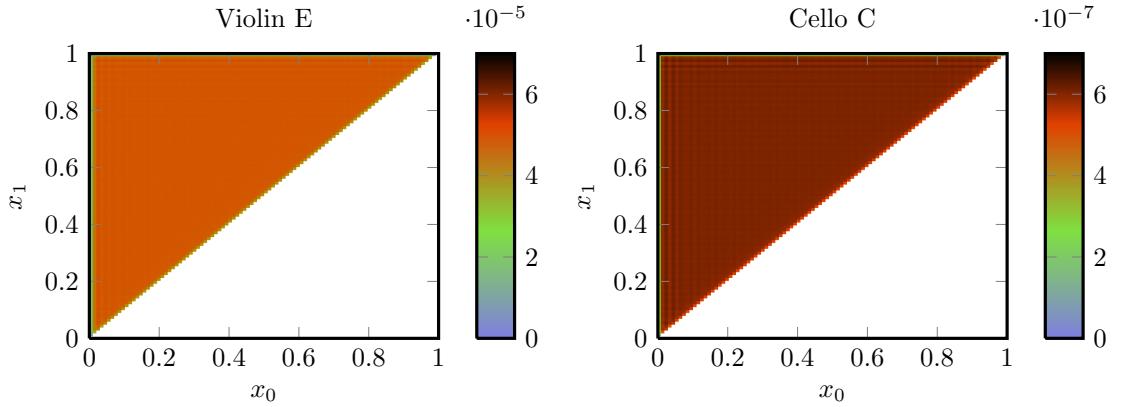


Figure A.2: $A_{11}B_{00} - A_{10}B_{01}$ for violin-E-I and cello-C-I. When $x_0 = 0$ or $x_1 = 0$, the expression has a value of 0; otherwise there is very little variation in its value.

In the real model, X_{3n} and Y_{3n} vary based on the mode, so (A.17) does not prove this conjecture. Figure A.1 suggests that X_{3n} and Y_{3n} decrease monotonically, but this depends on the precise modal decays r_n used. If r_n decreases abruptly as n increases, X_{3n} or Y_{3n} may increase.

Empirical plots of $A_{11}B_{00} - A_{10}B_{01}$ for $0 \leq x_0 < x_1 \leq 1$ are given in Figure A.2. Assuming that all other strings have similar A_{pq} and B_{pq} to these particular strings, these plots suggest that this conjecture is true for my physical constants.

Conjecture 2: $D_1 > 0$

As defined in Table 2.4,

$$D_1 = \frac{(B_{11}R_1 + A_{11}K_1 + 1)}{(B_{00}B_{11} - B_{01}B_{10})R_1 + (A_{11}B_{00} - A_{10}B_{01})K_1 + B_{00}} \quad (\text{A.18})$$

The numerator is positive since R_1 and K_1 are positive physical constants, and B_{11} and A_{11} have been shown to be positive (Lemma 2). The first term of the denominator has been shown to be positive (Lemma 4), the second term is assumed positive (Conjecture 1), and the third term has been shown to be positive (Lemma 2). Therefore, $D_1 > 0$ as required.

Appendix B

Performances of select sheet music

This appendix provides performances of four pieces of famous music for string ensembles.

W. A. Mozart: *Eine kleine Nachtmusik*

This is a string quartet for two violins, viola, and cello. The original version of this score came from the Mutopia¹ sheet music archive:

<http://www.mutopia-project.org/cgi-bin/piece-info.cgi?id=900>

The quartet was arranged by me to remove all double stops, as they are not yet supported by *Vivi*. I also added a few additional editorial dynamics and articulations. The resulting sheet music was processed with *Vivi* with no modifications to the audio output.

Eine kleine Nachtmusik
Movement 1

Wolfgang Amadeus Mozart (1756 - 1791)
arr. Graham Percival

Allegro

The musical score consists of four staves, each representing a different instrument: violin-1, violin-2, viola-1, and cello-1. The score is in common time, key signature of one sharp (F major), and tempo of 144 BPM. The arrangement removes double stops and includes editorial dynamics like f (fortissimo) and p (pianissimo). Articulations include slurs and grace notes.

Audio B.1: Eine kleine Nachtmusik

<http://percival-music.ca/dissertation/a.B.1.eine-kleine-nachtmusik.wav>

Music: <http://percival-music.ca/dissertation/eine-kleine-nachtmusik.pdf>

LilyPond: <http://percival-music.ca/dissertation/eine-kleine-nachtmusik.ly>

¹The Mutopia Project hosts music typeset with LilyPond which has been placed under the public domain or under the Creative Commons Attribution or Attribution-ShareAlike copyleft licenses.

J. S. Bach: *Concerto in D minor for two violins and strings*

This is a concerto for two violins accompanied by a small string orchestra. This version has a total of four violins, one viola, and one cello. The original version of this score came from the Mutopia sheet music archive:

<http://www.mutopiaproject.org/cgi-bin/piece-info.cgi?id=3>

I added some editorial dynamics and articulations. The resulting sheet music was processed with *Vivi* with no modifications to the audio output.

Concerto in D minor
for two violins and strings

Johann Sebastian Bach (1685-1750)
BWV 1043

Audio B.2: Bach double

<http://percival-music.ca/dissertation/a.B.2.bach-double.wav>

Music: <http://percival-music.ca/dissertation/bach-double.pdf>

LilyPond: <http://percival-music.ca/dissertation/bach-double.ly>

J. Pachelbel: *Canon in D*

This is a work for three violins and cello. The original version was from the Lyco² sheet music archive:

<http://sheetmusic.lyco.org.au/Pachelbel%20-%20Canon%20in%20D/>

The only modifications made by me were to add an initial dynamic for each part (*mp* for violins and *mp - p* for cello), and set the instrument names according to the format expected by *Vivi*. The resulting sheet music was processed with *Vivi* with no modifications to the audio output.

²Lyco is the Launceston Youth and Community Orchestra from Tasmania, Australia. After typesetting works for their own use, the members make their work available under the Creative Commons Attribution-ShareAlike 3.0 license.

Canon in D

Johann Pachelbel (1653-1706)

violin-1

violin-2

violin-3

cello-1

Audio B.3: Pachelbel

<http://percival-music.ca/dissertation/a.B.3.pachelbel-canon.wav>

Music: <http://percival-music.ca/dissertation/pachelbel-canon.pdf>

LilyPond: <http://percival-music.ca/dissertation/eine-kleine-nachtmusik.ly>

F. J. Gossec: *Gavotte*

This is the final piece of Suzuki Violin Book 1 (Suzuki 1978a). The typesetting was performed by me, with minor modifications to dynamics. The violin part was extracted and processed with *Vivi*. The piano part was extracted, a MIDI track was generated with LilyPond, and then audio was created with `timidity++` using the fluidr3_gm 3.1 soundfont. The two audio files were mixed after applying an amplification of +6 dB to the violin part. Since *Vivi* is only intended to perform stringed instrumental music, I feel the manual processing of the piano part was justified.

Gavotte

François-Joseph Gossec (1734 - 1829)

violin-1

piano

Audio B.4: Gavotte

<http://percival-music.ca/dissertation/a.B.4.suzuki-1-17-gavotte.wav>

Music: <http://percival-music.ca/dissertation/suzuki-1-17-gavotte.pdf>

LilyPond: <http://percival-music.ca/dissertation/eine-kleine-nachtmusik.ly>

Appendix C

Source code, raw data, and copyleft licenses

To encourage further modifications by other researchers, all source code and raw data is available under permissive copyleft licenses. The source code for Artifastring and *Vivi* is published under the GNU General Public License version 3 or higher. The full license is available at:

<http://www.gnu.org/licenses/>

This dissertation is published under the Creative Commons Attribution-ShareAlike 2.5 UK: Scotland. The full license is available at:

<http://creativecommons.org/licenses/by-sa/2.5/scotland/>

Physical modelling The source code for analysis of the experimental data, a python implementation of the physical model for easy experimentation, and the full C++ implementation of Artifastring:

<https://github.com/gperciva/artifastring>

An archival version of this repository at the time of this dissertation's publication:

<http://percival-music.ca/dissertation/artifastring-phd.tar.bz2>

The raw data for the modal decays and instrument body impulse responses are at:

<http://percival-music.ca/dissertation/physical-raw.html>

***Vivi* source code** The source code for the virtual musician and selected sheet music:

<https://github.com/gperciva/vivi>

An archival version of this repository at the time of this dissertation's publication:

<http://percival-music.ca/dissertation/vivi-phd.tar.bz2>

The raw data for the training datasets for *Vivi* are:

<http://percival-music.ca/dissertation/vivi-training-data.tar.bz2>

This dissertation The full source code for this L^AT_EX document, including all graphs and figures:

<http://percival-music.ca/dissertation/dissertation.tar.bz2>

Bibliography

- Abe, M. & Smith, J. O. (2004), ‘Design Criteria for Simple Sinusoidal Parameter Estimation Based on Quadratic Interpolation of FFT Magnitude Peaks’, *Audio Engineering Society Convention 117*.
- Adrien, J.-M. (1991), The missing link: modal synthesis, in G. De Poli, A. Piccialli & C. Roads, eds, ‘Representations of musical signals’, MIT Press, Cambridge, MA, USA, pp. 269–298.
- Almeida, A., Lemare, J., Sheahan, M., Judge, J., Auvray, R., Dang, K. S., John, S., Geoffroy, J., Katupitiya, J., Santus, P., Skougarevsky, A., Smith, J. & Wolfe, J. (2010), Clarinet parameter cartography: Automatic mapping of the sound produced as a function of blowing pressure and reed force, in ‘International Symposium on Music Acoustics’, Sydney and Katoomba, pp. 1–6.
- Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K., Mitchell, I., Plumley, M., Waugh, B., White, E. P., Wilson, G. & Wilson, P. (2012), ‘Best Practices for Scientific Computing’, *Computing Research Repository arxiv: abs/1210.0530v3*.
- Ashworth, J. (1997), The Naming of Hosts, Technical Report 2100, Internet Engineering Task Force.
URL: <http://www.ietf.org/rfc/rfc2100.txt>
- Beazley, D. (2003), ‘Automated scientific software scripting with SWIG’, *Future Generation Computer Systems* **19**(5), 599–609.
URL: <http://swig.org>
- Beazley, D. (2010), Understanding the python gil, in ‘PyCON’, Atlanta, Georgia.
- Berdahl, E., Niemeyer, G. & Smith, J. O. (2008), ‘Feedback Control of Acoustic Musical Instruments’, *Technical Report STAN-M-120* **120**.
- Bolanowski, S. J., Gescheider, G. A., Verrillo, R. T. & Checkosky, C. M. (1988), ‘Four channels mediate the mechanical aspects of touch’, *The Journal of the Acoustical Society of America* **84**(5), 1680–1694.
- Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992), A training algorithm for optimal margin classifiers, in ‘Proceedings of the fifth annual workshop on Computational learning theory’, COLT ’92, ACM, New York, NY, USA, pp. 144–152.
- Boutin, H. & Besnainou, C. (2008), ‘Physical parameters of the violin bridge changed by active control’, *The Journal of the Acoustical Society of America* **123**(5), 3656–3656.
- Box, G. E. P. (1979), Robustness in the Strategy of Scientific Model Building, in R. L. Launer & G. N. Wilkinson, eds, ‘Robustness in statistics’, Academic Press, London, UK, pp. 201–236.

- Brossier, P. (2006), Automatic Annotation of Musical Audio for Interactive Applications, PhD thesis, Queen Mary University of London, UK.
URL: <http://aubio.piem.org/phd/>
- Chafe, C. (1990), Pulsed noise in self-sustained oscillations of musical instruments , in 'International conference on Acoustics, Speech, and Signal Processing', pp. 1157–1160.
- Chafe, C. (1993), Tactile Audio Feedback, in 'International Computer Music Conference', Tokyo.
- Chan, L., Kirsop, B. & Arunachalam, S. (2011), 'Towards Open and Equitable Access to Research and Knowledge for Development', *PLoS Med* **8**(3).
- Chang, C.-C. & Lin, C.-J. (2011), 'LIBSVM: A library for support vector machines', *ACM Transactions on Intelligent Systems and Technology* **2**(3), 27–1.
URL: <http://www.csie.ntu.edu.tw/cjlin/libsvm>
- Charles, J. (2010), Playing Technique and Violin Timbre: Detecting Bad Playing, PhD thesis, Dublin Institute of Technology.
- Chia, Y. C., Hong, B. Y., Lee, C. H. & Lim, B. K. J. (2006), RoboFiddler, fourth-year final project, University of Adelaide, Australia.
- Chudy, M. & Dixon, S. (2012), Recognising Cello Performers using Timbre Models, Technical Report EECSRR-12-01, Queen Mary University of London, UK.
- Collins, C. D. (2009), Connecting Science and the Musical Arts in Teaching Tone Quality: Integrating Helmholtz Motion and Master Violin Teachers' Pedagogies, PhD thesis, George Mason University.
- Conklin, H. A. (1999), 'Generation of partials due to nonlinear mixing in a stringed instrument', *The Journal of the Acoustical Society of America* **105**(1), 536–545.
- Cook, P. R. (1995), A Hierarchical System for Controlling Synthesis by Physical Modeling, in 'International Computer Music Conference', Banff, Canada.
- Cook, P. R. (2002), *Real sound synthesis for interactive applications*, A K Peters.
- Cooley, J. & Tukey, J. (1965), 'An Algorithm for the Machine Calculation of Complex Fourier Series', *Mathematics of Computation* **19**(90), 297–301.
- Cortes, C. & Vapnik, V. (1995), 'Support-vector networks', *Machine Learning* **20**(3), 273–297.
- Cremer, L. (1984), *The physics of the violin*, MIT Press. Translated by John S. Allen.
- de Cheveigné, A. & Kawahara, H. (2002), 'YIN, a fundamental frequency estimator for speech and music', *Journal of the Acoustical Society of America* **111**(4), 1917–1930.
- Demoucron, M. (2008), On the control of virtual violins: Physical modelling and control of bowed string instruments, PhD thesis, IRCAM, Paris.
- Demoucron, M. & Rasamimanana, N. (2009), Score based real-time performance with a virtual violin, in 'International Conference on Digital Audio Effects', Como, Italy.

- Dooley, I. & Kale, L. (2006), Quantifying the Interference Caused by Subnormal Floating-Point Values, in 'Proceedings of the Workshop on Operating System Interference in High Performance Applications'.
- Ericsson, K. A., Krampe, R. T. & Tesch-Römer, C. (1993), 'The role of deliberate practice in the acquisition of expert performance', *Psychological Review* **100**(3), 363–406.
- Firth, I. (1985), 'Construction and performance of quality commercial violin strings', *Journal of the Catgut Acoustical Society* **44**, 17–20.
- Fitzgerald, B., Coates, J. & Lewis, S. (2007), *Open content licensing: cultivating the creative commons*, Sydney University Press.
- URL:** <http://creativecommons.org>
- Fletcher, H., Blackham, E. D. & Stratton, R. (1962), 'Quality of Piano Tones', *The Journal of the Acoustical Society of America* **34**(6), 749–761.
- Franchetti, F., Kral, S., Lorenz, J. & Ueberhuber, C. W. (2005), 'Efficient utilization of SIMD extensions', *Proceedings of the IEEE* **93**(2), 409–425.
- Friberg, A. (2006), 'pDM: An Expressive Sequencer with Real-Time Control of the KTH Music-Performance Rules', *Computer Music Journal* **30**(1), 37–48.
- Friberg, A., Colombo, V., Frydén, L. & Sundberg, J. (2000), 'Generating Musical Performances with Director Musices', *Computer Music Journal* **24**(3), 23–29.
- Friedman, M. (1937), 'The use of ranks to avoid the assumption of normality implicit in the analysis of variance', *Journal of the American Statistical Association* **32**(200), 675–701.
- Frigo, M. & Johnson, S. G. (2005), 'The Design and Implementation of FFTW3', *Proceedings of the IEEE* **93**(2), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".
- URL:** <http://libfftw.org>
- Fritz, C., Blackwell, A. F., Cross, I., Woodhouse, J. & Moore, B. C. J. (2012), 'Exploring violin sound quality: Investigating English timbre descriptors and correlating resynthesized acoustical modifications with perceptual properties', *The Journal of the Acoustical Society of America* **131**(1), 783–794.
- Fritz, C., Cross, I., Moore, B. C. & Woodhouse, J. (2007), 'Perceptual thresholds for detecting modifications applied to the acoustical properties of a violin', *J. Acoust. Soc. Am.* **122**, 3640–3650.
- Gargouri, Y., Hajjem, C., Larivière, V., Gingras, Y., Carr, L., Brody, T. & Harnad, S. (2010), 'Self-Selected or Mandated, Open Access Increases Citation Impact for Higher Quality Research', *PLoS ONE* **5**(10), e13636.
- Garritan Libraries (2012), 'Garritan personal orchestra 4', company website. Accessed 2012-Jan.
- URL:** <http://www.garritan.com/>
- Geringer, J. M. & Allen, M. L. (2004), 'An analysis of vibrato among high school and university violin and cello students', *Journal of Research in Music Education* **52**(2), 167–178.

- Goldberg, D. (1991), ‘What every computer scientist should know about floating-point arithmetic’, *ACM Comput. Surv.* **23**(1), 5–48.
- Good, M. (2001), ‘MusicXML: An internet-friendly format for sheet music’, *XML Conference and Expo* pp. 3–4.
- Guennebaud, G., Jacob, B. et al. (2010), ‘Eigen v3’.
URL: <http://eigen.tuxfamily.org>
- Guettler, K. (2002), The Bowed String. On the Development of Helmholtz Motion and On the Creation of Anomalous Low Frequencies, PhD thesis, KTH, Sweeden.
- Guettler, K. (2011), ‘How does rosin affect sound?’, *ASTA String Research Journal* **II**, 37–47.
- Guettler, K. & Askenfelt, A. (1995), Relation between bow resonances and the spectrum of a bowed string, in ‘International Symposium of Musical Acoustics’, Dourdan, France, pp. 232–237.
- Hamasaki, M., Takeda, H., Hope, T. & Nishimura, T. (2009), Network Analysis of an Emergent Massively Collaborative Creation Community: How Can People Create Videos Collaboratively without Collaboration?, in ‘International Conference on Weblogs and Social Media’, The AAAI Press.
- Hassaballah, M., Omran, S. & Mahdy, Y. B. (2008), ‘A review of SIMD multimedia extensions and their usage in scientific and engineering applications’, *The Computer Journal* **51**(6), 630–649.
- Helmholtz, H. L. F. (1895), *On the sensations of tone as a physiological basis for the theory of music*, 3rd edn, Longmans, Green, London. Alexander J. Ellis (translator).
- Hsu, C. W., Chang, C. C. & Lin, C. J. (2003), A practical guide to support vector classification, Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. Last updated 2010.
- Hutchins, C. M. (1981), ‘The Acoustics of Violin Plates’, *Scientific American* **245**(4), 170.
- IEEE Computer Society (2008), IEEE Standard for Floating-Point Arithmetic, Technical Report 754-2008, 3 Park Avenue, New York, NY 10016-5997, USA.
- Inácio, O., Antunes, J. & Wright, M. C. M. (2008), ‘Computational modelling of string–body interaction for the violin family and simulation of wolf notes’, *Journal of Sound and Vibration* **310**(1-2), 260–286.
- Intel (2007), *Intel SSE4 Programming Reference*.
- Jansson, E. (2002), Acoustics for violin and guitar makers, Technical report, KTH Department of Speech, Music, and Hearing, Sweeden.
URL: <http://www.speech.kth.se/music/acvigkeit4/>
- Jarvelainen, H., Välimäki, V. & Karjalainen, M. (2001), ‘Audibility of the timbral effects of inharmonicity in stringed instrument tones’, *Acoustics Research Letters Online* **2**(3), 79–84.
- Jones, E., Oliphant, T., Peterson, P. et al. (2001–), ‘SciPy: Open source scientific tools for Python’.
URL: <http://www.scipy.org/>

- Kajita, S., Nakano, T., Goto, M., Matsusaka, Y., Nakaoka, S. & Yokoi, K. (2011), VocaWatcher: Natural singing motion generator for a humanoid robot, in 'Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on', pp. 2000–2007.
- Karjalainen, M., Ansalo, P., Mäkivirta, A., Peltonen, T. & Välimäki, V. (2002), 'Estimation of Modal Decay Parameters from Noisy Response Measurements', *J. Audio Eng. Soc* **50**(11), 867–878.
- Katayose, H., Hashida, M., De Poli, G. & Hirata, K. (2012), 'On Evaluating Systems for Generating Expressive Music Performance: the Rencon Experience', *Journal of New Music Research* **41**(4), 299–310.
- Kenmochi, H. (2010), VOCALOID and Hatsune Miku phenomenon in Japan, in 'InterSinging 2010 - First Interdisciplinary Workshop on Singing Voice', Tokyo, Japan, pp. 1–4.
- Kenmochi, H. & Ohshita, H. (2007), VOCALOID – Commercial singing synthesizer based on sample concatenation, in 'Interspeech'.
- Kimura, M. (1999), 'How to Produce Subharmonics on the Violin', *Journal of New Music Research* **28**, 178–184.
- Kirke, A. & Miranda, E. R. (2009), 'A survey of computer systems for expressive music performance', *ACM Computing Surveys (CSUR)* **42**(1), 3.
- Klapuri, A. & Davy, M. (2006), *Signal Processing Methods for Music Transcription*, Springer, New York.
- Kruskal, W. H. & Wallis, W. A. (1952), 'Use of ranks in one-criterion variance analysis', *Journal of the American statistical Association* **47**(260), 583–621.
- Kusuda, Y. (2008), 'Toyota's violin-playing robot', *Industrial Robot* **35**(6), 504–506.
- Laakso, M., Welling, P., Bukvova, H., Nyman, L., Björk, B.-C. & Hedlund, T. (2011), 'The Development of Open Access Journal Publishing from 1993 to 2009', *PLoS ONE* **6**(6), e20961.
- Lattner, C. (2005), Macroscopic Data Structure Analysis and Optimization, PhD thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL.
URL: <http://llvm.org>
- Lee, N., Smith, J. O. & Välimäki, V. (2010), 'Analysis and Synthesis of Coupled Vibrating Strings Using a Hybrid Modal-Waveguide Synthesis Model', *Audio, Speech, and Language Processing, IEEE Transactions on* **18**(4), 833–842.
- Lessig, L. (2001), *The Future of Ideas: The Fate of the Commons in a Connected World*, Random House, New York.
- Li, L. & Lin, H.-T. (2007), Ordinal Regression by Extended Binary Classification, in B. Schölkopf, J. Platt & T. Hoffman, eds, 'Advances in Neural Information Processing Systems 19', MIT Press, Cambridge, MA, pp. 865–872.
- Lindemann, E. (2007), 'Music Synthesis with Reconstructive Phrase Modeling', *Signal Processing Magazine, IEEE* **24**(2), 80–91.

- Maestre, E. (2009), Modeling instrumental gestures: an analysis/synthesis framework for violin bowing, PhD thesis, Universitat Pompeu Fabra.
- McIntyre, M. E., Schumacher, R. & Woodhouse, J. (1983), 'On the oscillations of musical instruments', *Journal of the Acoustical Society of America* **74**(5).
- Minsky, M. (1961), 'Steps toward Artificial Intelligence', *Proceedings of the IRE* **49**(1), 8–30.
- Miranda, E. R., Magee, W. L., Wilson, J. J., Eaton, J. & Palaniappan, R. (2011), 'Brain-Computer Music Interfacing (BCMI)', *Music and Medicine* **3**(3), 134–140.
- Morrison, J. D. & Adrien, J.-M. (1993), 'MOSAIC: A Framework for Modal Synthesis', *Computer Music Journal* **17**(1), –45.
- Nakano, T. & Goto, M. (2011), Vocalistener2: A singing synthesis system able to mimic a user's singing in terms of voice timbre changes as well as pitch and dynamics, in 'Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on', pp. 453–456.
- Nethercote, N. & Seward, J. (2007), Valgrind: a framework for heavyweight dynamic binary instrumentation, in 'Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation', PLDI '07, ACM, New York, NY, USA, pp. 89–100.
- URL:** <http://valgrind.org>
- Nichols, C. S. (2003), The vBow: An Expressive Musical Controller Haptic Human-Computer Interface, PhD thesis, Stanford University.
- Peeters, G. (2004), A large Set of Audio Features for Sound Description (Similarity and Classification) in the CUIDADO project, Technical report.
- Penttinen, H., Pakarinen, J., Välimäki, V., Laurson, M., Li, H. & Leman, M. (2006), 'Model-based sound synthesis of the guqin', *The Journal of the Acoustical Society of America* **120**(6), 4052–4063.
- Pérez, A. (2009), Enhancing Spectral Synthesis Techniques with Performance Gestures using the Violin as a Case Study, PhD thesis, Universitat Pompeu Fabra.
- Pérez, A. & Wanderley, M. M. (2012), Learning and extraction of violin instrumental controls from audio signal, in 'Proceedings of the second international ACM workshop on Music information retrieval with user-centered and multimodal strategies', MIRUM '12, ACM, New York, NY, USA, pp. 25–30.
- Pickering, N. (1985), 'Physical properties of violin strings', *Journal of the Catgut Acoustical Society* **44**, 6–8.
- Project Petrucci LLC (2012), 'International Music Score Library Project'. Accessed 02 June 2012.
URL: <http://www.imslp.org/>
- Raman, C. V. (1920), 'Experiments with Mechanically-played Violins', *Proceedings of the Indian Association of Cultivation of Science* **6**, 19–36.

- Rasamimanana, N. H. (2008), Geste instrumental du violoniste en situation de jeu: analyse et modélisation, PhD thesis, Université Paris 6 - IRCAM UMR STMS. (in French).
- Rossing, T. D. (2010), *Science of String Instruments*, Springer.
- Saygin, A. P., Cicekli, I. & Akman, V. (2000), ‘Turing test: 50 years later’, *Minds and Machines* **10**(4), 463–518.
- Scavone, G. P. & Cook, P. R. (2005), ‘Rtmidi, Rtaudio, and a synthesis toolkit (STK) update’, *Proceedings of the 2005 International Computer Music Conference* pp. 1–4.
URL: <https://ccrma.stanford.edu/software/stk/>
- Schelleng, J. C. (1973), ‘The bowed string and the player’, *Journal of the Acoustical Society of America* **53**(1), 26–41.
- Schoner, B., Cooper, C., Douglas, C. & Gershenfeld, N. (1999), ‘Data-Driven Modeling of Acoustical Instruments’, *Journal of New Music Research* **28**(2), 81–89.
- Schoonderwaldt, E. (2009), Mechanics and acoustics of violin bowing: Freedom, constraints and control in performance, PhD thesis, KTH, Sweden.
- Schwarz, D. (2007), ‘Corpus-Based Concatenative Synthesis’, *Signal Processing Magazine, IEEE* **24**(2), 92–104.
- Serafin, S. (2004), The sound of friction: real-time models, playability and musical applications, PhD thesis, CCRMA, Standford University.
- Serra, X. (1989), A System for Sound Analysis/Transformation/Synthesis based on a Deterministic plus Stochastic Decomposition, PhD thesis, Stanford University.
- Sinclair, S., Florens, J.-L. & Wanderley, M. M. (2010), A Haptic Simulator for Gestural Interaction with the Bowed String, in ‘Proceedings of the Congrès Fra cais d’Acoustique’, Lyons, France.
- Smith, J. H. & Woodhouse, J. (2000), ‘The tribology of rosin’, *Journal of the Mechanics and Physics of Solids* **48**, 1633–1681.
- Smith, J. O. (1992), ‘Physical Modeling Using Digital Waveguides’, *Computer Music Journal* **16**(4), –74.
- Smith, J. O. (2010), *Physical Audio Signal Processing*, online book, accessed 2012-Jan.
URL: <http://ccrma.stanford.edu/~jos/pasp/>
- Smith, J. O. (2011), *Spectral Audio Signal Processing*, online book, accessed 2012-Jan.
URL: <http://ccrma.stanford.edu/~jos/sasp/>
- Solis, J., Takanishi, A. & Hashimoto, K. (2010), Development of an Anthropomorphic Saxophone-Playing Robot, in J. Angeles, B. Boulet, J. Clark, J. Kövecses & K. Siddiqi, eds, ‘Brain, Body and Machine’, Vol. 83, Springer Berlin / Heidelberg, chapter Advances in Intelligent and Soft Computing, pp. 175–186.

- Solis, J., Taniguchi, K., Ninomiya, T., Petersen, K., Yamamoto, T. & Takanishi, A. (2009), 'Implementation of an Auditory Feedback Control System on an Anthropomorphic Flutist Robot Inspired on the Performance of a Professional Flutist', *Advanced Robotics* **23**(14), 1849–1871.
- Stallman, R. (2010), *Free Software, Free Society: Selected Essays of Richard M. Stallman*, Free Software Foundation, Inc.
- Stallman, R. et al. (2012), *Using and Porting the GNU Compiler Collection*, Free Software Foundation.
URL: <http://gcc.gnu.org/>
- Sterling, M. (2010), Empirical Physical Modeling Methods for Bowed-String and Wind Instruments , PhD thesis, University of Rochester.
- Stockham, T. G. (1966), High-speed convolution and correlation, in 'Proceedings of the April 26–28, 1966, Spring joint computer conference', AFIPS '66 (Spring), ACM, New York, NY, USA, pp. 229–233.
- Suzuki, S. (1978a), *Violin Part Volume 1*, Summy-Birchard Music.
- Suzuki, S. (1978b), *Violin Part Volume 2*, Summy-Birchard Music.
- Suzuki, Y. & Takeshima, H. (2004), 'Equal-loudness-level contours for pure tones', *The Journal of the Acoustical Society of America* **116**, 918.
- Türckheim, F., Smit, T., Hahne, C. & Mores, R. (2010), Novel Impulse Response Measurement Method for Stringed Instruments, in 'The 20th International Congress on Acoustics'.
- Turing, A. M. (1950), 'Computing machinery and intelligence', *Mind* **59**(236), 433–460.
- Tushnet, R. (2007), 'Payment In Credit: Copyright Law And Subcultural Creativity', *Law and Contemporary Problems* **70**.
- Tzanetakis, G. (2002), Manipulation, Analysis and Retrieval Systems for Audio Signals, PhD thesis, Princeton University, NJ, USA.
- Tzanetakis, G. (2007), Marsyas: a case study in implementing Music Information Retrieval Systems, in S. Shen & L. Cui, eds, 'Intelligent Music Information Systems: Tools and Methodologies', Information Science Reference.
- Tzanetakis, G., Benning, M. S., Ness, S. R., Minifie, D. & Livingston, N. (2009), Assistive music browsing using self-organizing maps, in 'International Conference on Pervasive Technologies Related to Assistive Environments', ACM International Conference Proceeding Series, ACM.
- Vamvakousis, Z. (2011), The EyeHarp: A Gaze-Controlled Musical Instrument, Master's thesis, Universitat Pompeu Fabra, Barcelona.
- Vandewalle, P., Kovacevic, J. & Vetterli, M. (2009), 'Reproducible research in signal processing', *Signal Processing Magazine, IEEE* **26**(3), 37–47.
- Vienna Symphonic Library GmbH (2012), 'Vienna instruments', company website. Accessed 2012-Jan.
URL: <http://www.vsl.co.at/>

- Warwick, K. (1989), *Control Systems an Introduction*, Series in Systems and Control Engineering, Prentice Hall International, Hertfordshire, UK.
- Widmer, G., Flossmann, S. & Grachten, M. (2009), 'YQX plays Chopin', *AI Magazine* **30**(3), 35.
- Woodhouse, J. (2004), 'Plucked Guitar Transients: Comparison of Measurements and Synthesis', *Acta Acustica united with Acustica* **90**(5), 945–965.
- Woodhouse, J. (2005), 'On the "bridge hill" of the violin', *Acustica - Acta Acustica* **91**, 155–165.
- Woodhouse, J. & Galluzzo, P. M. (2004), 'The Bowed String As We Know It Today', *Acta Acustica united with Acustica* **90**, 579–589.
- Woodhouse, J. & Loach, A. R. (1999), 'Torsional Behaviour Of Cello Strings', *Acta Acustica united with Acustica* **85**(5), 734–740.
- Woodhouse, J., Manuel, E. K. Y., Smith, L. A., Wheble, A. J. C. & Fritz, C. (2012), 'Perceptual Thresholds for Acoustical Guitar Models', *Acta Acustica united with Acustica* **98**(3), 475–486.
- Wright, M. (2008), The Shape of an Instant: Measuring and Modeling Perceptual Attack Time with Probability Density Functions, PhD thesis, Stanford University, CA, USA.
- Wrzeciono, P. & Marasek, K. (2010), Violin Sound Quality: Expert Judgements and Objective Measurements, in Z. Ras & A. Wieczorkowska, eds, 'Advances in Music Information Retrieval', Vol. 274, Springer Berlin / Heidelberg, pp. 237–260.
- Wu, Y., Kuvinchkul, P., Cheung, P. & Demiris, Y. (2010), 'Towards Anthropomorphic Robot Thereminist', pp. 235–240.
- Wyse, L., Nanayakkara, S., Seekings, P., Ong, S. H. & Taylor, E. (2012), Perception of vibrotactile stimuli above 1kHz by the hearing-impaired, in 'New Interfaces for Musical Expression'.
- Young, D. S. (2007), A methodology for investigation of bowed string performance through measurement of violin bowing technique, PhD thesis, MIT.
- Ziegler, J. G. & Nichols, N. B. (1942), 'Optimum settings for automatic controllers', *trans. ASME* **64**(11).