From Maxima Basics to Research Calculations

A guided reading of twelve quantum-physics programs

Learn the Maxima evaluation model, symbolic calculus, complex arithmetic, matrices, numerical methods, and regression testing before moving through the project scripts from approachable scalar models to an inverse-scattering capstone.

Open-Problem computational companion

Maxima 5.49.0 baseline – July 2026

Preface

This book has one practical purpose: make every program in `maxima/` readable to someone who is migrating to Maxima from hand calculation, another computer-algebra system, or a numerical language. The programs are small, but their ideas are not. They mix exact algebra, floating computation, complex branches, matrix construction, asymptotics, and physical diagnostics. Reading them in filename order would begin with an advanced WKB recurrence and hide the language lessons inside the physics.

Part I therefore builds the Maxima evaluation model before the case studies introduce scalar functions, matrices, complex branches, asymptotics, and discretized inverse problems in increasing mathematical depth.

The twelve drivers are executable research baselines. They do not claim to be production solvers or to close the associated open research problems. Their tests protect internal identities and numerical regressions; they are not a substitute for convergence, conditioning, uncertainty, or independent physical validation.

About this book

This is an authored computational textbook and code-reading companion. It is intended for readers who know undergraduate calculus and linear algebra but may be new to computer algebra, shell workflows, or research-grade numerical checks. Part I supplies the Maxima language and workflow foundation. The remaining parts use complete, executable physics programs as progressively more demanding case studies.

Learning map

Part	Main skill	Programs reached in that part
Part I	Foundation: evaluation, functions, lists, symbolic calculus, complex values, matrices, numerics, and tests	The language used by every driver
Part II	Scalar models and exact identities	`05_nonmarkovian_zeno.mac`, `07_orbital_free_dft.mac`, `06_efimov_corrections.mac`
Part III	Small matrices, poles, fields, and affine propagation	`09_tensor_force_eft.mac`, `10_differentiable_scattering.mac`, `08_supercritical_dirac.mac`, `11_structured_light.mac`, `12_accelerated_quantum_dynamics.mac`
Part IV	Indexed asymptotics and coupled spectral calculations	`01_exact_wkb.mac`, `03_threshold_universality.mac`, `04_floquet_dirac_comb.mac`
Part V	Capstone: discretized inverse problem and a custom solver	`02_nonhermitian_inverse_scattering.mac` and `lib/02_marchenko_numeric.mac`

Contents

Abbreviations and recurring symbols

Term	Meaning
CAS	Computer-algebra system
DFT	Density-functional theory
EFT	Effective field theory
SI	International System of Units
SOC	Spin–orbit coupling
WKB	Wentzel–Kramers–Brillouin asymptotic method
E1, E2, M1	Electric-dipole, electric-quadrupole, and magnetic-dipole amplitudes
`%i`	Maxima's imaginary unit
`fpprec`	Number of decimal digits used for bigfloat arithmetic
`;` / `$`	Evaluate with / without automatic display

List of Figures

List of Tables

List of Listings

Part I

The Maxima on-ramp

Chapter 1

Starting Maxima: shell, prompts, and clean sessions

Abstract. This chapter starts with the smallest useful mental model of Maxima, then establishes the complete path from installation and a terminal prompt to a reproducible calculation. It distinguishes shell commands from Maxima statements, explains interactive labels, creates a first saved program, and introduces the clean-session policy used by every driver.

Keywords. Maxima Shell, Batch Mode, Input Labels, Exact Arithmetic, Help System, Reproducibility

What Maxima does

Maxima is a computer-algebra system: it reads a mathematical expression, constructs a symbolic object, evaluates what is known, simplifies the result, and optionally displays it. Its first advantage over an ordinary calculator is that numbers need not be rounded and names need not already have numerical values.

```
(%i1) 2 + 3;
(%o1) 5
(%i2) 1/3 + 1/6;
(%o2) 1/2
(%i3) x + x;
(%o3) 2*x
(%i4) x + 1;
(%o4) x + 1
```

The last line is not a failure. Because no value has been assigned to `x`, Maxima keeps it as a symbol. This ability to preserve exact numbers and unknown parameters is what later makes differentiation, equation solving, limits, and parameter-dependent matrices possible.

Interactive transcript notation. Maxima prints an input prompt such as `(%i1)` and creates output labels such as `(%o1)`. Only the expression following an input prompt, including its semicolon or dollar sign, is entered by the user; the displayed labels are not input. The input label `%i1` is distinct from the imaginary unit `%i` because it contains a trailing digit.

Keep the terminal and Maxima languages separate

The terminal and Maxima are two different interpreters. A terminal command starts programs or moves between directories. A Maxima statement represents a calculation. Plotting adds a third layer because Maxima normally asks gnuplot to draw the figure.

Table 1.1: Three layers in the command-line workflow

Layer	Typical input	Purpose
Terminal or shell	`cd`, `pwd`, `maxima -version`	locate files and start a program

Maxima	`2+3;`, `f(x):=x^2$`	construct and evaluate mathematical objects
Gnuplot backend	launched by `plot2d`	display a plot or write a graphics file

If Maxima reports a syntax error for `cd` or the shell reports "command not found" for `2+3;`, the input was sent to the wrong layer.

Install and confirm the command-line setup

Use the current instructions on the official Maxima installation page [1]. The shortest common routes are summarized here; the official page remains authoritative when an operating system changes.

Table 1.2: Practical installation routes

System	Route
macOS	With Homebrew, run `brew install maxima`. MacPorts and graphical interfaces are also documented officially.
Windows	Use the current 64-bit installer from the Maxima project. The native installation includes command-line and graphical interfaces; a POSIX-compatible shell such as WSL or Git Bash is needed for the supplied `.sh` runners.
Linux	Use the distribution package or the current official package. Check the reported version because a distribution repository may contain an older release.

Open a terminal and change to the workspace root, the directory that contains both `maxima/` and `Book/`. Replace the example path below by the location of this project. The following are shell commands; do not type a displayed shell prompt itself.

```
cd /path/to/Open-Problem
pwd
ls maxima Book
command -v maxima
maxima --version
maxima --help
```

`pwd` prints the current directory, while `ls maxima Book` confirms that both project directories are visible. If `command -v maxima` prints an executable path, the terminal can find Maxima. The project baseline used for this manuscript is Maxima 5.49.0. A different recent version may work, but record it with every numerical result because simplification, special-function, and floating output can change between releases.

Start and leave an interactive session

Start the console with one shell command:

```
maxima
```

Maxima then displays input labels `(%i1)`, `(%i2)`, and so on. A semicolon evaluates an expression and prints a correspondingly numbered output label. The symbol `%` recalls the most recent output, while a label such as `%o1` recalls a specific earlier result.

```
(%i1) 2 + 3;
(%o1) 5
(%i2) %^2;
```

```
(%o2) 25
(%i3) %o1 + 10;
(%o3) 15
(%i4) quit();
```

History labels depend on the order of an interactive session: editing or inserting one line changes their meaning. Saved programs in this book use descriptive names instead.

Save and run the smallest program

A `.mac` file is plain text containing ordinary Maxima statements. In a text editor, save the following as `first_calculation.mac` in the workspace root:

```
/* first_calculation.mac */
exact_value : 1/3 + 1/6$
decimal_value : float(sqrt(8))$
print("exact =",exact_value)$
print("decimal =",decimal_value)$
```

The colon gives a name to a value, the dollar sign evaluates without automatic display, and `print` selects the results that form the report. These ideas are developed carefully in the next chapter. For now, run the file from the terminal:

```
maxima --batch=first_calculation.mac
```

The report contains the exact value $1/2$ and a decimal approximation to $\sqrt{8}$. This establishes the complete minimal workflow: edit plain text, save it, ask Maxima to evaluate it, and read the resulting report.

Run one supplied research program

From the workspace root, the shortest project command is

```
maxima --batch=maxima/05_nonmarkovian_zeno.mac
```

This command establishes that the supplied file can be found and executed. Its scalar formulas and Maxima constructs are explained before the case-study chapter.

For reproducible logs, use the same explicit flags as the supplied runner:

```
maxima --very-quiet --no-init --suppress-input-echo --quit-on-error \
  --batch=maxima/05_nonmarkovian_zeno.mac
```

The option `-b` is the short form of `-batch`. Maxima reads the file, evaluates its statements in order, and exits. The shell status is zero after success and nonzero after a failure when `-quit-on-error` is used. Immediately after a run, inspect it with

```
echo $?
```

Here the shell expands `$?` to the exit status of the immediately preceding command. A displayed 0 means that the process completed successfully; any other integer signals failure.

Run the complete project suites

The supplied shell scripts resolve the workspace root and start every file in a fresh Maxima process:

```
./maxima/run_models.sh
./maxima/run_tests.sh
./Book/book.sh verify
```

The first command executes all twelve model drivers. The second executes the paired regression tests. The third runs the tests, rebuilds this book, and checks that the PDF and auxiliary files are in their intended locations.

Figure 1.1 separates the shell orchestration layer from the Maxima calculation it launches. This distinction explains why a printed error and a nonzero process status are both important.

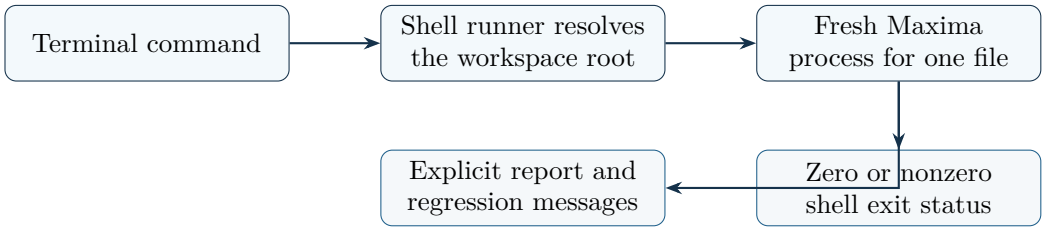

Figure 1.1: Execution path for the supplied model and test runners. Each input file receives a clean Maxima process, while the shell collects human-readable output and a machine-readable exit status.

Comments, help, and discoverability

Maxima comments begin with `/*` and end with `*/`; they may span several lines. Use them to record units, assumptions, branch choices, and the reason for a tolerance. The interactive help system is searchable:

```
describe(diff);
example(integrate);
apropos("matrix");
```

`describe` opens reference material, `example` shows executable examples, and `apropos` searches names. The versioned Maxima manual documents the syntax of the installed system; another CAS may use different conventions [2].

Two statement terminators

A semicolon `;` evaluates and displays the result. A dollar sign `$` evaluates but suppresses automatic display. Both store the result; only the display behavior differs.

```
a : 7/20;       /* Maxima displays 7/20 */
b : 4$          /* Maxima displays nothing */
print(a*b)$     /* print deliberately: 7/5 */
```

Research scripts normally use `$` for intermediate objects and explicit `print` or `printf` statements for stable, readable output. This is why a batch run prints a compact report instead of every large symbolic matrix.

When a result or input looks wrong

Maxima distinguishes an unknown symbol from an error. An unassigned name stays in the result; a malformed statement produces a syntax message. Inspect the current user state before assuming the algebra is wrong:

```
a : 3$
f(x) := a*x + 1$
values;             /* user variables, including a */
functions;          /* user-defined functions, including f(x) */
kill(a)$             /* remove one definition */
f(2);                /* 2*a + 1: a is symbolic again */
kill(all)$           /* remove every user definition */
```

In an interactive session, correct a syntax error and enter the statement again. In a batch calculation, treat the first error as the cause: later expressions may depend on an object that was never created. The flag `-quit-on-error` enforces that rule and turns the failure into a nonzero shell status.

The common clean-session header

Nearly every driver begins with the following pattern:

```
kill(all)$
display2d:false$
ratprint:false$
linel:200$
```

`kill(all)`
: removes user definitions so a previous interactive session cannot silently change the model. It is destructive to the current Maxima session, so save interactive work first.

`display2d:false`
: selects one-line output suited to logs and shell redirection.

`ratprint:false`
: suppresses warnings about internal rational conversions that would otherwise obscure the report.

`linel:200`
: gives the printer a wider line before it wraps.

Batch flags and the working directory

A batch program is a saved sequence of ordinary Maxima statements. The robust command shown earlier uses flags with specific jobs. `-no-init` prevents a personal initialization file from altering the run. `-suppress-input-echo` keeps the report compact. `-very-quiet` suppresses banners and expression labels, while explicit `print` and `printf` calls remain visible. `-quit-on-error` turns a Maxima error into a nonzero shell status instead of continuing through a broken calculation.

Several files use relative paths, including the Model 2 helper path. Their working directory is the workspace root. The supplied shell runners change to that root before Maxima starts; the relative helper path is

```
maxima/lib/02_marchenko_numeric.mac
```

Constants and names

Built-in constants begin with a percent sign: `%pi`, `%e`, `%i`, and `inf`. Maxima is case-sensitive, so `E`, `e`, and `%e` are different. Project names such as `Gamma`, `Sigma0`, and `LambdaStar` retain their case.

References

[1] *Maxima Installation*. Maxima Project. 2026. URL: https://maxima.sourceforge.io/download.html (visited on 07/17/2026).

[2] Maxima Project. *Maxima 5.49.0 Manual*. Maxima Project. 2025. URL: https://maxima.sourceforge.io/docs/manual/index.html (visited on 07/17/2026).

Chapter 2

Expressions, assignment, functions, and evaluation

Abstract. Maxima's evaluator is easiest to control when expressions, assignments, delayed functions, equations, and quoted names are treated as different objects. This chapter develops that distinction, introduces the expression-tree model and operator grammar, and explains when exact values are converted to floating approximations.

Keywords. Expression Trees, Assignment, Delayed Functions, Substitution, Quoting, Exact Arithmetic

The read–evaluate–simplify cycle

For every statement, Maxima first reads the syntax, then replaces names that have values and evaluates function calls, then applies automatic simplification. The final semicolon or dollar sign controls only whether the result is displayed; it does not turn evaluation on or off.

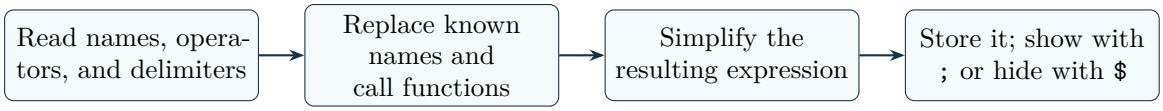

Figure 2.1: The evaluation cycle used by both interactive statements and saved programs.

This model explains two common surprises. An unassigned name remains symbolic rather than causing an error, while a name with an old assignment is replaced even if the user intended to treat it as a fresh symbol.

Values, functions, and equations are different objects

The operators `:`, `:=`, and `=` look similar but create different kinds of object.

Table 2.1: Assignment, function-definition, and equation syntax

Form	Meaning	Example
`x : value`	evaluate now and assign a value	`a : 4/5$` stores an exact rational
`f(x) := body`	define a function evaluated when called	`f(x) := a*x^2$` uses the current value of `a`
`left = right`	construct an equation or replacement rule	`x^2=2` is data for a solver

The function argument `x` is local to each call; other names in its body are looked up when the function is called. Compare an immediate value with a delayed function:

```
a : 2$
saved_value : a^2$
f(x) := a*x^2$
a : 3$
[saved_value, f(2)];        /* [4,12] */
```

`saved_value` was computed when assigned. The later call `f(2)` sees the new value of `a`. This is why an editable parameter block near the top of a driver controls functions defined below it.

Operators, grouping, and compound statements

Powers bind more tightly than multiplication, which binds more tightly than addition. Parentheses make denominators, signs, and compound conditions explicit. Square brackets construct lists; parentheses group expressions and can also form a comma-separated compound statement whose value is its final expression.

```
value : (a:2, b:3, a^2 + b^2)$   /* value is 13 */
scaled : (x + y)/(1 + x^2)$
flag : (x > 0) and (x < 1)$
```

Comparisons use `=`, `#`, `<`, `<=`, `>`, and `>=`; logical expressions use `and`, `or`, and `not`. An equation made with `=` is symbolic data unless it appears in a context that explicitly asks for a truth value.

Multiplication must be written with `*`: `x*y` is a product, while `xy` is one variable name. Functions require parentheses, as in `sin(x)` rather than handwritten-style `sin x`. Newlines are only whitespace; a semicolon or dollar sign ends the statement.

```
/* Explicit multiplication, powers, and function calls. */
f(x) := 2*x^2 + exp(-a*x)$
```

Named numerical options often use equation syntax. For example, a `quad_qags` call may include the option `epsrel=1.0e-8`. Such an option does not globally assign `epsrel`; the receiving function interprets the equation locally.

Expressions have structure

Maxima does not store `a*x^2+b*x+c` as a line of text. It stores an operator with arguments, and each argument may be another expression. After the basic evaluation rules are clear, the inspection functions `op`, `args`, and `part` expose that tree:

```
expr : sin(x)^2 + a*x$
op(expr);              /* + */
length(args(expr));    /* 2 */
part(expr,1);          /* one stored term */
freeof(y,expr);        /* true */
```

Canonical simplification may reorder commutative terms, so `part(expr,1)` is not a stable way to identify a term by mathematical meaning. Use `coeff`, substitution, or a predicate when structure rather than display position matters.

Substitution and temporary evaluation

For the expression below, both replacements are visible in the result:

```
kill(all)$
expr : a*x^2 + b$
subst(3,x,expr);              /* 9*a + b */
ev(expr,x=3,a=2,b=1);         /* 19 */
expr;                         /* still a*x^2 + b */
```

Neither operation permanently changes `expr`. `subst` emphasizes a replacement; `ev` evaluates an existing object under a temporary set of values. The project uses both common forms of `subst`:

```
subst([E=Ep, g=gv], Dexpr)   /* several equation-style replacements */
subst(Ep, E, Dexpr)          /* new value, old symbol, expression */
```

The evaluator `ev` combines substitution with optional evaluation flags:

```
ev(S[n], x=log(rprobe/L))
ev(kappaNLO(n)-kappaLO(n), re=0, kso=0)
```

These longer forms are the same ideas applied to named model expressions.

Quoting and inert names

A leading apostrophe prevents one evaluation step. A minimal example makes the difference visible:

```
x : 5$
[x,'x];              /* [5,x] */
kill(x)$
```

In the project, `lam:'lam` ensures that a characteristic-polynomial variable is a fresh symbolic name, and `matrixmap('float,M)` passes the name of the function `float` to a higher-order operation.

Quoting is not decoration. Without it, a global value accidentally assigned to `lam` could be substituted before `charpoly` receives the symbol.

Exact and floating values

Keep the symbolic object until a numerical approximation is actually needed:

```
exact_value : sqrt(8);          /* 2*sqrt(2) */
float(exact_value);             /* 2.82842712474619 */
```

- `7/20`, `1/9`, and `sqrt(5)` are exact.
- `0.35`, `1e-5`, and `1.44316060e-25` are machine floats.
- `float(expr)` requests ordinary floating evaluation.
- `bfloat(expr)` requests arbitrary-precision bigfloat evaluation, controlled by the global precision variable `fpprec`.

The notation `1e-5` means the machine float 10^{-5}; it is not exact. Bigfloat output uses a `b` exponent marker. Chapter 8 shows why converting an exact rational to a bigfloat preserves information that was already lost in a machine decimal.

Do symbolic work before inserting decimals whenever practical. Once an approximate coefficient enters a determinant, simplifier and equality results are also approximate.

Maxima does not infer physical units from a bare number. A value such as `length:2.0$` has no built-in knowledge of metres or nanometres. The drivers therefore declare one consistent unit convention in their parameter blocks and use names, comments, and table headers to preserve it. Dimensional consistency remains a mathematical check the author must perform.

Choosing a simplifier

There is no universal "simplify" command. The project chooses operations matched to the algebra:

`expand`
: distribute products and powers; essential before extracting polynomial coefficients.

`ratsimp`
: canonicalize rational expressions.

`radcan`
: simplify radicals and rational powers.

`trigsimp`
: apply trigonometric identities.

`rectform`
: rewrite complex expressions into real and imaginary parts before extracting them.

The operation chosen is part of the mathematical claim. A test based on `radcan` is asserting a radical identity; a floating residual is making a different, weaker assertion.

Chapter 3

Algebra, equations, assumptions, and solving

Abstract. This chapter supplies the algebraic operations that readers often expect from a computer-algebra system but that are easy to misuse when migrating from another tool. It distinguishes transformation from proof, equations from truth values, exact solving from numerical root finding, and declared assumptions from facts Maxima can infer automatically.

Keywords. Algebraic Forms, Equation Solving, Assumptions, Predicates, Exact Tests, Residuals

Transformations answer different questions

Equivalent expressions can have very different useful forms. Choose the form that exposes the quantity you need:

```
p : (x - 1)*(x + 2)$
expand(p);                          /* x^2 + x - 2 */
factor(x^2 + x - 2);                /* (x - 1)*(x + 2) */
ratsimp((x^2 - 1)/(x - 1));         /* x + 1, generically */
num((x + 1)/(x^2 + 1));
denom((x + 1)/(x^2 + 1));
partfrac(1/(x*(x + 1)),x);
```

The simplification of $(x^2 - 1)/(x - 1)$ is valid away from $x = 1$. A CAS normally returns a generic expression; it does not automatically preserve the excluded point of the original formula. Record domain restrictions whenever cancellation changes them.

Equations are symbolic objects

The expression `x^2=4` is an equation. It is neither an assignment nor a Boolean result. Follow one equation through its complete lifecycle: construct it, solve it, then substitute each proposed solution back into the original expression.

```
poly : x^2 - 5*x + 6$
solutions : solve(poly = 0,x);      /* two equations: x=2 and x=3 */
rhs(solutions[1]);                  /* extract one proposed value */
ev(poly,solutions[1]);              /* 0: its exact residual */
```

The order of returned roots is not a mathematical promise, so select by value or property rather than assuming the first root is always the smaller one. For several equations, the corresponding operations are

```
linsolve([2*x + y = 1, x - y = 0],[x,y]);  /* [x=1/3,y=1/3] */
eliminate([y = x^2, y = 2*x + 3],[y]);
```

`solve` targets algebraic equations, `linsolve` exploits linear structure, and `eliminate` removes named variables. Larger polynomial systems may require `algsys`. A returned solution must still

be checked in the original equations, because clearing denominators or squaring radicals can introduce extraneous roots.

Symbolic solving and numerical root finding differ

Exact solving attempts to describe all algebraic solutions. Numerical root finding searches for one solution in a supplied region. The two operations answer different questions:

```
solve(x^2 - 2 = 0,x);             /* exact roots */
root : find_root(cos(x) - x,x,0,1)$
abs(float(cos(root) - root));     /* numerical residual */
```

The bracket in `find_root` is part of the numerical argument. Substitution of the returned root into the original expression yields its residual.

Assumptions are part of the calculation

Integrals and simplifications can depend on sign, reality, or integrality. Make those facts explicit and remove temporary facts when they are no longer valid:

```
assume(a > 0)$
integrate(exp(-a*x),x,0,inf);
facts();
forget(a > 0)$

declare(n,integer)$
properties(n);
remove(n,integer)$
```

An assumption is global state. This is one reason reproducible drivers begin with `kill(all)` and tests run in fresh processes.

Truth can be true, false, or unknown

The literal values are `true` and `false`, but a symbolic comparison may be undecidable with the known assumptions. The third possible result is `unknown`:

```
is(x > 0);                        /* unknown without an assumption */
is(0.2 > 0);                      /* true */
[numberp(1/3),numberp(1.0),numberp(%pi),numberp(x)];
/* [true,true,false,false] */
```

`numberp` asks whether an expression is explicitly numeric; symbolic constants such as `%pi` are not machine numbers. An `if` condition must resolve to `true` or `false`; `unknown` is not permission to guess a branch. Add assumptions, substitute numerical values, or report that the assertion was not established.

```
if is(equal(ratsimp((x + 1)^2 - (x^2 + 2*x + 1)),0))
then print("identity verified")
else error("identity not established")$
```

For a floating calculation, replace symbolic equality by an absolute-plus- relative residual test:

```
residual : abs(float(computed - expected))$
scale : max(1.0,abs(float(expected)))$
if not is(residual < 1.0e-10*scale) then
  error("numerical residual",residual)$
```

This exact-versus-numerical distinction is the basis of every paired regression file in the project.

Chapter 4

Lists, control flow, local scope, and generated objects

Abstract. Lists, loops, conditions, and local blocks form the programming grammar behind the project tables and tests. This chapter explains one-based indexing, safe list construction, the complete loop and conditional patterns used by the drivers, anonymous functions, generated matrices, and indexed symbolic objects.

Keywords. Lists, Control Flow, Local Scope, Map, Makelist, Anonymous Functions, Indexed Objects

Lists are one-based

```
energies : [-1/2,-7/20,-1/5,-1/10]$
first_energy : energies[1]$
last_energy : energies[length(energies)]$
```

Maxima list and matrix indices start at 1. This matters in expressions such as `u[a][b,1]`: select list element `a`, then matrix row `b`, column 1.

Common nonmutating list operations are

```
first(energies)$
rest(energies)$
length(energies)$
cons(-1,energies)$
endcons(0,energies)$
append([a,b],[c,d])$
sort([3,1,2]);          /* [1,2,3] */
```

Assignment to `energies[2]` changes the existing list. Constructing a new list leaves the original baseline value available to later calculations. The integer helpers `floor(5/2)` and `mod(5,2)` return 2 and 1; later they recover grid and channel indices from a flattened index.

Conditions and truth values

An `if` expression returns the selected branch and therefore works inside an assignment as well as at statement level:

```
sign_class(x) := if x > 0 then "positive"
                 elseif x < 0 then "negative"
                 else "zero"$

safe_ratio(a,b) := if is(equal(b,0)) then false else a/b$
```

The condition must evaluate to `true` or `false`. If symbols remain and Maxima cannot decide, it reports an undetermined condition. Add assumptions or return the unevaluated case explicitly; do not silently guess.

Map, construct, or loop

Begin with one function and one list so the three programming forms can be compared directly:

```
square(x) := x^2$
numbers : [1,2,3]$
map(square,numbers);                  /* [1,4,9] */
makelist(square(j),j,1,3);            /* [1,4,9] */
for j in numbers do print(square(j))$ /* prints 1, 4, and 9 */
```

Use `map` to apply a function to an existing list, `makelist` to construct a new list from an index or range, and a `for` loop for repeated assignment or output. The corresponding project idioms are

```
action_table : map(radial_action,energies)$
qs : makelist(q, q, -Q, Q)$
pole_scan : makelist([kp,cabs(determinant(Finv(kp)))],kp,pole_grid)$
for n:0 thru 3 do print([n,float(kappaNLO(n))])$
for lam in lambda_grid do
  for tv in time_grid do print([lam,tv,float(coherence(lam,tv))])$
```

The main loop forms are worth reading literally:

```
for item in data do print(item)$
for n:1 thru 5 do print(n^2)$
for j thru 3 do print(j)$       /* omitted start means 1 */
for n:5 step -1 thru 1 do print(n)$
for x:1 next 2*x while x < 20 do print(x)$
for n:1 thru 3 do (s:s+n^2, print(n,s))$
```

The default start and step are 1, so `for j thru 3` abbreviates `for j:1 step 1 thru 3`. The `next` form states a custom update rule. The parenthesized body groups several comma-separated statements into one loop body. The loop control variable is localized and its previous value is restored afterward. Assignments to other names in the body, such as `s`, affect the surrounding scope unless those names are local in an enclosing `block`.

Local variables with block

Without a local declaration, assignment normally affects a global name. A `block` creates local variables and returns the value of its last expression:

```
row(x) := block([y:x^2],[x,y])$
row(3);                          /* [3,9] */

observable_row(k) := block([S:Smat(k), tau:mean_delay(k)],
  [k,cabs(determinant(S)),realpart(tau),imagpart(tau)])$
```

In the first function, `y` exists only during the call. In the project form, `S` and `tau` are likewise local. The list on the last line is the function result. Local initialization after a colon is evaluated when the block begins.

Anonymous functions and generated matrices

Start with an explicit diagonal example:

```
d : [2,3,5]$
genmatrix(lambda([i,j],kron_delta(i,j)*d[i]),3,3);
/* matrix([2,0,0],[0,3,0],[0,0,5]) */
```

`kron_delta(i,j)` is 1 when the two indices are equal and 0 otherwise. The `lambda` expression is an anonymous two-argument function; `genmatrix` calls it for every row and column. This

pattern creates diagonal matrices, Toeplitz blocks, correlation matrices, discretized operators, and conversion layers without spelling out each entry.

Indexed symbolic variables

`Q[0]`, `Q[1]`, and `S[n]` are subscripted variables, not list elements unless their base name was explicitly assigned a list. Model 1 uses them as a dictionary of WKB coefficients:

```
S[0] : sqrt(Q[0])$
S[1] : -diff(S[0],x)/(2*S[0])$
for n:2 thru Nwkb do S[n] : ratsimp(/* recurrence */)$
```

Chapter 5

Symbolic calculus, series, and transforms

Abstract. Differentiation, exact integration, ordinary differential equations, limits, Taylor expansions, sums, products, and integral transforms are the symbolic backbone of the model derivations and regression identities. This chapter emphasizes expression form, assumptions, noun results, and the transition from a formal result to a checked numerical calculation.

Keywords. Differentiation, Exact Integration, Differential Equations, Limits, Taylor Series, Symbolic Sums, Laplace Transform

Differentiate an expression, not a black box

`diff(expr,x)` differentiates an expression with respect to `x`. Start with derivatives whose output can be verified immediately:

```
diff(x^3,x);                    /* 3*x^2 */
diff(sin(x),x);                 /* cos(x) */
diff(x^3,x,2);                  /* 6*x */
```

The third argument is the derivative order. The pole-sensitivity program then uses the same operation on a named model expression and stores two derivatives once:

```
DEexpr : diff(D(E,g),E)$
Dgexpr : diff(D(E,g),g)$
```

It later substitutes a pole and a coupling. This is both faster and clearer than reconstructing the symbolic derivatives inside every loop iteration.

Higher derivatives may be requested as `diff(f(x),x,2)`. In radial problems, the three-dimensional Laplacian of a radial function is

$$\nabla^2 n(r) = n''(r) + \frac{2}{r} n'(r).$$

At $r = 0$, symbolic simplification or a limit exposes whether the apparent singularity is removable before numerical substitution.

Exact integration

Indefinite and definite integration use the same function with different numbers of arguments:

```
integrate(x^2,x);               /* x^3/3 */
integrate(x^2,x,0,1);           /* 1/3 */
integrate(exp(-x),x,0,inf);     /* 1 */
```

An indefinite result omits the arbitrary integration constant; add it when writing the general antiderivative. Maxima can also return exact angular and improper integrals. Model 3 computes Gaunt coefficients from

$$\frac{\sqrt{(2\ell+1)(2\ell'+1)}}{2} \int_{-1}^{1} P_\ell(z) P_2(z) P_{\ell'}(z)\, dz,$$

while the DFT test checks normalization through

```
integrate(4*%pi*r^2*nrad(r),r,0,inf)
```

An unevaluated `integrate(...)` result is not automatically an error. It may mean that assumptions are missing or that the integral is not elementary. The project chooses inputs for which the required exact results are available.

A first ordinary differential equation

Derivatives inside an equation must remain symbolic until an ODE solver sees them. A leading apostrophe keeps `diff` in that noun form:

```
ode : 'diff(y,x) = -k*y$
general_solution : ode2(ode,y,x);
initial_solution : ic1(general_solution,x=0,y=y0);
```

`ode2` returns an equation for `y`; `ic1` applies one initial condition to a first-order solution. Substitution of that solution into the differential equation produces a residual. Higher-order and coupled systems may require other solvers or numerical integration, but the object pattern is the same: an equation enters the solver and a solution equation is returned.

Limits and removable singularities

Model 4 writes $\sin(ks)/k$ through a limit:

```
sine_kernel(k,s) := block([x],limit(sin(x*s)/x,x,k))$
```

At $k = 0$, the limit is s; using the raw quotient would produce $0/0$. Model 5 uses a different kind of limit: a scaled strong-coupling limit with `lam` approaching `inf`. Both cases express the mathematics more faithfully than adding an arbitrary small denominator.

Series and asymptotic information

A Taylor series is a local representation, not a global identity. Maxima keeps the truncation order as part of the result:

```
taylor(log(1 + x),x,0,5)$
taylor(exp(-lambda*t),t,0,4)$
```

`ratdisrep` converts the result to an ordinary polynomial expression and therefore discards the explicit series-order marker. A series is specified by its expansion point, retained order, and parameter regime. Model 5's Zeno limit applies the same structure to a rescaled variable rather than through a direct Taylor call.

Transforms, sums, and products

The first three arguments of `laplace` are the expression, original variable, and transform variable:

```
laplace(exp(-t),t,z);            /* 1/(z + 1) */
```

The regression suite applies the same operation to `laplace(sigma2*exp(-t/tc),t,z)` to recover a rational memory kernel. Finite symbolic sums build WKB recurrences and structured fields:

```
sum(k^2,k,1,n)$
product(1 + a*k,k,1,n)$
sum(Q[m]*S[n-m],m,0,n)$
```

If the upper limit is symbolic, Maxima may return a closed form, a conditional result, or an unevaluated sum. Small integer substitutions expose the first terms of a formal recurrence. The

next chapter treats the named special functions that often remain after calculus and transforms have done all useful simplification.

Chapter 6

Special functions for physics

Abstract. Special functions are the natural solution language of linear differential equations, boundary-value problems, transforms, and symmetry decompositions. This chapter organizes the families most often encountered in quantum and mathematical physics, gives their Maxima names, connects them to physical equations, and explains recurrence relations, defining differential equations, branch structure, and numerical precision.

Keywords. Gamma Function, Error Function, Bessel Functions, Hankel Functions, Legendre Polynomials, Laguerre Polynomials, Airy Functions, Hypergeometric Form, Elliptic Integrals

The families are organized by the mathematical structures that produce them: differential equations, basis expansions, transforms, and limiting forms.

A named function is often the simplest exact answer

The solution of a differential equation need not be elementary to be exact. Bessel functions encode cylindrical and radial waves, Legendre functions encode angular separation, Hermite and Laguerre polynomials encode oscillator and Coulomb bases, Airy functions resolve a simple turning point, and hypergeometric functions unify many of these families. Replacing such a function by a decimal too early discards recurrence relations, analytic continuation, and parameter dependence.

The installed help index exposes the Maxima names and conventions:

```
apropos("bessel")$
describe(bessel_j)$
example(legendre_p)$
```

Argument order and normalization conventions differ across software. The Maxima conventions are documented in the installed manual [1]; the low-order values below make those conventions explicit.

Gamma, beta, and Gaussian tails

The Gamma function continues the factorial through $\Gamma(n+1) = n!$, while the beta integral satisfies

$$B(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}.$$

Maxima keeps exact special values when it can:

```
gamma(1/2);                /* sqrt(%pi) */
beta(1/2,1/2);             /* %pi */
gamma(6)/gamma(5);         /* 5 */
```

The error functions `erf(x)` and `erfc(x)` describe integrated Gaussian weight, diffusion fronts, and normal-distribution tails. For a large positive argument, computing `1-erf(x)` can lose every significant digit. The directly evaluated `erfc(x)` represents the small complementary tail without that subtraction.

Bessel families and radial waves

The ordinary Bessel equation

$$x^2y'' + xy' + (x^2 - \nu^2)y = 0$$

has solutions J_ν and Y_ν, entered as `bessel_j(nu,x)` and `bessel_y(nu,x)`. Modified radial problems use `bessel_i` and `bessel_k`; spherical waves use `spherical_bessel_j` and `spherical_bessel_y`.

```
recurrence_residual : ratsimp(diff(bessel_j(nu,x),x)
  - (bessel_j(nu-1,x)-bessel_j(nu+1,x))/2)$
recurrence_residual;              /* 0 */
bessel_j(0,0);                    /* 1 */
```

For complex arguments, branch choices matter for the second-kind and modified families. At large magnitude, higher-precision and scaled forms avoid some machine underflow and overflow regimes.

Angular momentum and orthogonal polynomials

Maxima provides `legendre_p(l,z)` and `assoc_legendre_p(l,m,z)` for central-potential angular structure. Model 3 integrates products of Legendre polynomials to form exact Gaunt coefficients. A low-order identity is an effective convention check:

```
p2 : expand(legendre_p(2,z))$
ratsimp(p2 - (3*z^2 - 1)/2);      /* 0 */
```

The one-dimensional harmonic oscillator uses `hermite(n,x)`. Radial oscillator and Coulomb problems use `laguerre(n,x)` and `gen_laguerre(n,alpha,x)`. Orthogonality integrals require the correct weight and domain; an unweighted integral is not an orthogonality test.

```
hermite(2,x);                     /* 4*x^2 - 2 */
gen_laguerre(2,1,x)$
```

A normalized physics wavefunction contains scale factors and weight functions in addition to the library polynomial.

Airy functions at a turning point

Linearizing a potential near a simple classical turning point reduces the wave equation to Airy's equation. Maxima names its two independent solutions `airy_ai(x)` and `airy_bi(x)`, with derivatives `airy_dai(x)` and `airy_dbi(x)`. The decaying solution on the positive real axis is $\mathrm{Ai}(x)$; $\mathrm{Bi}(x)$ grows.

```
ai0 : airy_ai(0)$
residual : ratsimp(diff(airy_ai(x),x,2) - x*airy_ai(x))$
```

This local uniform representation explains why ordinary WKB coefficients become singular at a turning point. Model 1 brackets turning points but does not construct the Airy matching solution.

Hypergeometric form and analytic continuation

The syntax `hypergeometric(a_list,b_list,z)` represents ${}_pF_q$, with numerator parameters in the first list and denominator parameters in the second. Empty lists are meaningful. Model 12 uses

```
Cfun(t,G) := hypergeometric([], [1/2], -G*t^2/4)$
Sfun(t,G) := t*hypergeometric([], [3/2], -G*t^2/4)$
Kfun(t,G) := t^2/2*hypergeometric([1], [3/2,2], -G*t^2/4)$
```

These entire representations continue cosine-like and sine-over-root combinations smoothly through $G = 0$ and to negative G, avoiding a piecewise formula with an apparent division by zero. An elementary rewrite is useful only when it improves the intended limit or numerical regime.

Hankel functions and radiating waves

The combinations

$$H_\nu^{(1)}(z) = J_\nu(z) + iY_\nu(z), \qquad H_\nu^{(2)}(z) = J_\nu(z) - iY_\nu(z)$$

separate the two oscillatory directions of a cylindrical wave. With the time convention $e^{-i\omega t}$, $H_\nu^{(1)}(kr)$ has the large-radius phase of an outgoing wave and $H_\nu^{(2)}(kr)$ that of an incoming wave. Changing the time convention reverses this interpretation. Maxima provides both cylindrical and spherical forms:

```
outgoing_cyl : hankel_1(nu,k*r)$
incoming_cyl : hankel_2(nu,k*r)$
outgoing_sph : spherical_hankel1(l,k*r)$
incoming_sph : spherical_hankel2(l,k*r)$
```

The second-kind component carries the same branch structure as `bessel_y`. Consequently a complex radial continuation involves both the asymptotic radiation convention and the branch of the argument.

Elliptic integrals and nonlinear periods

The complete elliptic integrals in Maxima use the parameter m:

$$K(m) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - m\sin^2\theta}}, \qquad E(m) = \int_0^{\pi/2} \sqrt{1 - m\sin^2\theta}\, d\theta.$$

They are entered as `elliptic_kc(m)` and `elliptic_ec(m)`. Some references instead write a modulus k, in which case $m = k^2$. At $m = 0$, both functions equal $\pi/2$.

For a pendulum of length L, gravitational acceleration g, and angular amplitude θ_0, the exact period is

$$T(\theta_0) = 4\sqrt{\frac{L}{g}}\, K\left(\sin^2\frac{\theta_0}{2}\right).$$

The corresponding Maxima expressions and small-amplitude endpoint are

```
period(theta0) := 4*sqrt(L/g)
  *elliptic_kc(sin(theta0/2)^2)$
elliptic_kc(0);                 /* %pi/2 */
2*%pi*sqrt(L/g);               /* theta0 -> 0 period */
```

The incomplete first-, second-, and third-kind integrals are represented by `elliptic_f`, `elliptic_e`, and `elliptic_pi`. Their argument and parameter conventions are distinct from the complete forms and are stated in the Maxima manual [1].

References

[1] Maxima Project. *Maxima 5.49.0 Manual*. Maxima Project. 2025. URL: https://maxima.sourceforge.io/docs/manual/index.html (visited on 07/17/2026).

Chapter 7

Complex arithmetic and branch choices

Abstract. Complex-valued physics requires more than attaching the symbol `%i`. This chapter explains rectangular form, conjugation and modulus, principal logarithm and square-root branches, continuity along parameter paths, and the difference between a displayed principal value and a physically chosen Riemann sheet.

Keywords. Complex Arithmetic, Rectangular Form, Complex Modulus, Principal Branch, Branch Cuts, Analytic Continuation

Enter and inspect a complex number

Maxima's imaginary unit is `%i`; the ordinary name `i` has no special meaning unless the user assigns one. Explicit multiplication is still required:

```
z : 3 + 4*%i$
[realpart(z),imagpart(z),conjugate(z),cabs(z)];
/* [3,4,3-4*%i,5] */
rectform(exp(%i*%pi));              /* -1 */
```

The conjugate reverses the sign of the imaginary part, while `cabs` returns the nonnegative modulus. For a matrix, applying `transpose` alone does not conjugate its entries; a Hermitian adjoint needs both operations.

Rectangular form before component extraction

For a nontrivial expression z, a reliable pattern is

```
q : rectform(z)$
[realpart(q),imagpart(q),cabs(q)]$
```

`rectform` pushes complex exponentials, radicals, and products toward $a + ib$. `cabs` is the complex modulus. The generic `abs` is also used in the project, but `cabs(rectform(...))` makes intent explicit in branch-sensitive diagnostics.

Principal branches

Maxima uses principal branches for complex `sqrt` and `log`. Thus `sqrt(delta-k^2)` and `log(-%i*k)` do not by themselves track a chosen scattering sheet. A parameter scan that crosses a branch cut can jump even when the physical continuation is smooth.

Figure 7.1 shows the geometry behind the principal logarithm $\log z = \log|z| + i\arg z$. Points immediately above and below the negative real axis have arguments approaching π and $-\pi$, so a pointwise scan across the cut produces a jump of nearly $2\pi i$.

The jump can be seen directly by approaching -1 from above and below:

```
delta : 1.0e-6$
theta_above : float(imagpart(rectform(log(-1 + %i*delta))))$
theta_below : float(imagpart(rectform(log(-1 - %i*delta))))$
[theta_above,theta_below,theta_above-theta_below];
```

```
/* approximately [3.14159,-3.14159,6.28318] */
```

Nothing singular happened to either sampled point; the discontinuity comes from choosing principal arguments on opposite sides of the conventional cut.

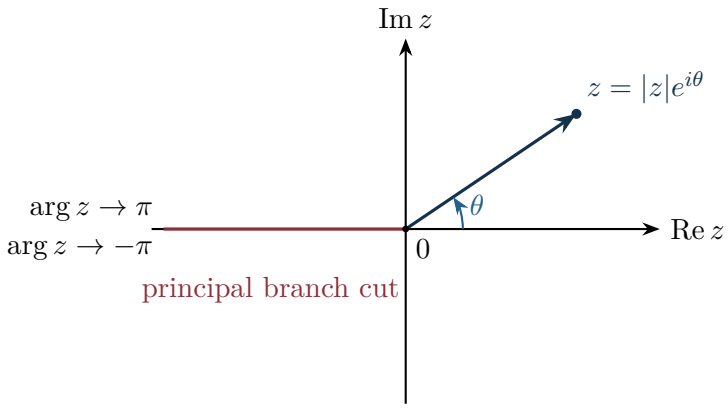

Figure 7.1: Principal-branch convention with $-\pi < \arg z \le \pi$. The negative real axis is the conventional cut for the principal logarithm and square root.

A coarse grid of principal-branch values does not perform analytic continuation. A physical continuation instead specifies a sheet and a continuous parameter path.

Chapter 8

Numerical precision, roots, quadrature, and convergence

Abstract. The numerical stage of a symbolic calculation introduces finite precision, conditioning, tolerances, convergence, and algorithm-specific status information. This chapter develops machine and arbitrary precision, bracketed and polynomial roots, adaptive quadrature, finite differences, residual scaling, and convergence studies used throughout the model suite.

Keywords. Machine Precision, Bigfloat Arithmetic, Root Finding, Quadrature, Finite Differences, Residual Tests, Convergence

Machine floats and bigfloats

Machine floats typically carry about sixteen decimal digits. Bigfloats use the decimal precision requested through `fpprec`:

```
fpprec : 40$
bfloat(1/10);       /* 40 digits from an exact rational */
bfloat(0.1);        /* promotes a value already rounded as a machine float */
```

Maxima writes bigfloat exponents with `b`; for example, `1.23b0` means 1.23×10^0. The marker identifies arbitrary-precision output and is not a variable name.

Increasing `fpprec` cannot recover information already lost in a decimal literal or an ill-conditioned subtraction. Exact integers, rationals, and symbolic constants preserve information until conversion. Changing precision affects arithmetic performed after the change; it does not add digits to an already computed result.

Bracketed roots

A complete numerical root calculation is

```
f(x) := cos(x) - x$
[f(0.0),f(1.0)];                    /* opposite signs */
root : find_root(f(x),x,0.0,1.0)$
[root,abs(float(f(root)))];          /* root and residual */
```

The interval is mathematical information: it states where a sign-changing root is expected. The last value is the residual of the equation passed to the solver. Model 1 uses the same form with `find_root(realpart(Qfull(rr,en)),rr,0.001,0.1)`. Fixed brackets can fail or select a different root when parameters move or merge turning points.

Adaptive quadrature

`quad_qags` returns a list rather than only the integral:

```
ans : quad_qags(sin(x),x,0,%pi,epsrel=1.0e-10)$
ans;
/* [value, absolute-error estimate, evaluations, status] */
```

For this example the value is approximately 2 and the status is 0. The first entry is the integral, so Model 1 uses `first(ans)`. The remaining entries carry the error estimate, evaluation count, and algorithm status; a nonzero status signals that the requested quadrature conditions were not met.

Polynomial roots

`allroots(p)` numerically solves a polynomial and returns equations. `rhs` extracts their numerical right-hand sides:

```
p : x^2 + 1$
map(rhs,allroots(p));                /* the two roots +%i and -%i */
```

Model 4 applies the same pattern to a characteristic polynomial. This compact method suits small matrices but can be ill-conditioned at larger dimension; it is not a replacement for a stable eigensolver and convergence study.

Finite differences and residuals

The centered difference

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

has truncation error of order h^2, but very small h amplifies floating cancellation. Model 10 compares this approximation with an analytic implicit derivative. Model 2 uses a centered difference to reconstruct a potential.

The following values show the competition, with exact target $\cos(0) = 1$:

```
f(x) := sin(x)$
for h in [0.1,0.01,0.001] do block([estimate],
  estimate:(f(h)-f(-h))/(2*h),
  print([h,estimate,abs(estimate-1.0)]))$
```

The error first decreases approximately as h^2. Extending the table to extremely small `h` eventually exposes floating cancellation.

A scale-aware numerical condition has the form

$$|r| \leq \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}}|q|.$$

The project tests use fixed tolerances because their baseline scales are known.

Convergence under refinement

A tighter tolerance changes one algorithmic request; it does not by itself establish convergence. Precision, root bracket, quadrature tolerance, grid spacing, cutoff, truncation order, and finite-difference step are independent numerical parameters. Convergence is the limiting behavior of an observable under refinement of the parameters on which its discretization depends. Competing truncation, roundoff, and conditioning errors can prevent monotone improvement.

Chapter 9

Scientific plots and publication figures

Abstract. A scientific figure is a quantitative argument with a visual encoding. This chapter connects Maxima's plotting interface to gnuplot, distinguishes exploratory display from publication output, demonstrates vector export and explicit data files, and develops standards for units, uncertainty, accessible design, captions, and reproducibility.

Keywords. Scientific Plotting, Gnuplot, Vector Graphics, Data Export, Grayscale Design, Uncertainty, Figure Captions

The plotting backend and a first curve

Maxima normally sends plotting instructions to gnuplot. Its terminal commands for locating the backend and reporting its version are

```
command -v gnuplot
gnuplot --version
```

The corresponding minimal Maxima plot is

```
plot2d(sin(x),[x,-%pi,%pi])$
```

The first argument is the expression and the second gives the variable and its domain. The final dollar sign suppresses textual output in Maxima; it does not suppress the graph. An absent graph window or graphics file indicates that the Maxima–gnuplot connection did not complete.

Interactive and publication output

An interactive plot exposes domains, roots, singularities, and scales. A publication figure adds a fixed physical size, axis quantities and units, curve encodings that survive grayscale conversion, and an explicit representation of uncertainty when uncertainty is part of the result.

Maxima's direct interface accepts function and discrete data together:

```
data : makelist([tau,exp(-tau)],tau,0,3,0.25)$
plot2d([exp(-tau^2/2),[discrete,data]],[tau,0,3],
  [style,lines,linespoints],
  [point_type,diamond],
  [legend,"Gaussian baseline","Exponential samples"],
  [xlabel,"scaled time tau"],
  [ylabel,"coherence"])$
```

In `makelist(expr,index,start,end,step)`, the fifth argument is the step. In `plot2d`, the first argument lists the two plotted objects, `[discrete,data]` marks tabulated coordinate pairs, `[tau,0,3]` sets the domain, and each remaining bracketed list is an option. Thus this example combines exponential samples with the continuous Gaussian curve used in Figure 9.1. Axis strings can contain the mathematical symbol and unit defined in the text.

Export vector line art

Vector PDF or SVG preserves curves, markers, text, and axes without resolution-dependent line edges. Gnuplot's `pdfcairo` terminal creates a fixed-size PDF directly from Maxima; terminal-specific options are documented by the gnuplot project [1]:

```
plot2d([exp(-tau^2/2),[discrete,data]],[tau,0,3],
  [style,lines,linespoints],
  [point_type,diamond],
  [legend,"Gaussian baseline","Exponential samples"],
  [xlabel,"scaled time tau"],
  [ylabel,"coherence"],
  [gnuplot_term,
    "pdfcairo enhanced color font 'Helvetica,9' size 8.6cm,6.2cm"],
  [gnuplot_out_file,"coherence.pdf"],
  [run_viewer,false])$
```

With `run_viewer:false`, Maxima writes `coherence.pdf` in its current working directory without opening a viewer.

The width 8.6 cm is an illustrative single-column target; a publisher's specified dimensions replace it in a submitted figure. Raster formats remain appropriate for microscopy, density maps, and other pixel data; their effective resolution depends on the final physical size.

Separate numerical data from rendering

`with_stdout` writes a plain numerical table that can be rendered by a separate plotting program. The first argument `true` in `printf(true,...)` means the current standard output; inside `with_stdout` that output is redirected to the named file.

```
with_stdout("coherence.dat",(
  printf(true,"# tau  coherence~%"),
  for row in data do
    printf(true,"~12,6e  ~12,6e~%",row[1],row[2])))$
```

The header defines the columns. Units, parameter values, precision, and a data-generation version can also be encoded in comment lines. Maxima preserves generated gnuplot instructions through `gnuplot_script_file` when rendering is performed outside the interactive session.

A publication-style visual grammar

Figure 9.1 uses both line style and marker shape, so its two curves remain distinct without color. The legend names the mathematical roles rather than "curve 1" and "curve 2," and the axes give symbols and dimensionless units.

Uncertainty and convergence encodings

Error bars encode pointwise uncertainty, credible bands encode an interval at each abscissa, and refinement envelopes encode variation under numerical resolution. A smooth interpolating line contains no uncertainty information unless one of these encodings accompanies it. Logarithmic axes, excluded or singular regions, and truncated axis ranges change the quantitative reading of a figure and therefore appear explicitly in its axes or caption.

References

[1] Gnuplot Development Team. *Gnuplot Documentation*. Gnuplot Project. URL: https://gnuplot.info/documentation.html (visited on 07/17/2026).

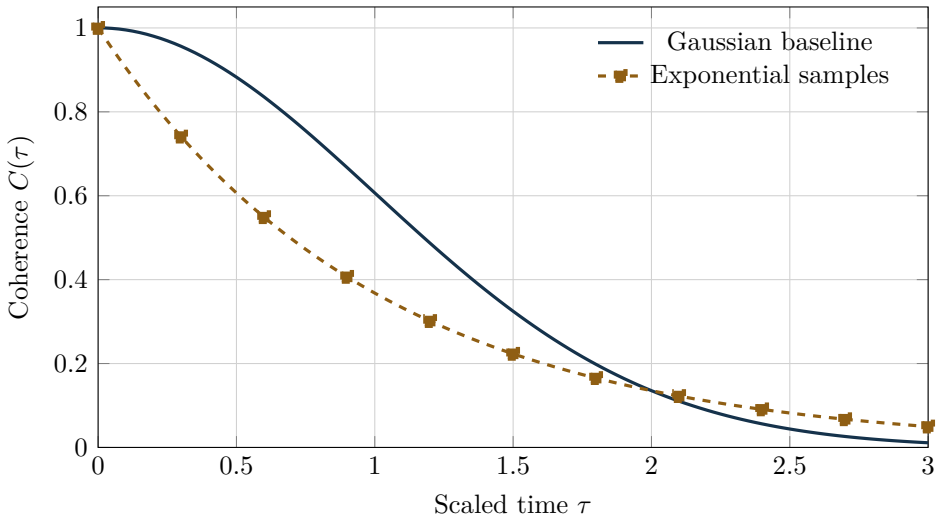

Figure 9.1: Illustrative publication-style comparison. Solid versus dashed lines and square markers remain distinguishable in grayscale.

Chapter 10

Matrices, vectors, and linear systems

Abstract. Matrices in Maxima are explicit algebraic objects with one-based indices and a noncommutative product. This chapter develops construction, dimension tracking, block assembly, transpose and adjoint distinctions, determinants and linear solves, covariance propagation, and the numerical limitations of explicit inversion.

Keywords. Matrix Construction, Matrix Product, Block Matrices, Linear Systems, Covariance, Conditioning

Construction and indexing

Start with a matrix and column vector containing ordinary numbers:

```
M : matrix([2,1],[1,3])$
v : matrix([1],[2])$
M[2,1];                          /* 1 */
[length(M),length(M[1])];        /* [2,2]: rows and columns */
```

Rows are written as lists and indexed from 1. Thus `M[i]` is row `i`, while `M[i,j]` is one entry. The form `M[i,j]:value` mutates that entry. A row vector is `matrix([1,2])`; the two one-element rows in `matrix([1],[2])` create the column vector above. Common constructors are

```
symbolic_M : matrix([a,b],[c,d])$
I2 : ident(2)$
Z23 : zeromatrix(2,3)$
```

`genmatrix` constructs an array from an entry rule. `submatrix(3,M3,3)` deletes row 3 and column 3; it does not select them. That detail explains how Model 3 retains the leading 2×2 block of a 3×3 Gaunt matrix.

The dot operator

Matrix multiplication uses `.`. Continue the numerical example before reading the project notation:

```
M.v;                             /* matrix([4],[7]) */
transpose(v).M.v;                /* 18 */

Hflow : U.H0.transpose(U)$
scalar : transpose(psi).J.psi$
```

The operator `*` denotes ordinary commutative multiplication. It is fine for a scalar times a matrix, but it is not the operator for matrix products.

Transpose is not an adjoint

`transpose(M)` does not conjugate complex entries. It equals the Hermitian adjoint only when the entries are real. Model 9 is intentionally real, so its orthogonal rotations use transpose. A complex generalization must apply conjugation as well.

```
adjoint(A) := conjugate(transpose(A))$
```

Block assembly

Model 4 defines

```
join4(A,B,C,D) := addrow(addcol(A,B),addcol(C,D))$
```

`addcol(A,B)` joins matrices horizontally; `addrow` then stacks the two block rows. Matrix products are defined only for compatible inner dimensions. With Sambe cutoff Q, `Ns=2*Q+1`; every transfer matrix is $2N_s \times 2N_s$.

Determinants, inverses, and solves

For the matrix `M` defined above, both operations are exact:

```
determinant(M);                 /* 5 */
invert(M);                      /* matrix([3/5,-1/5],[-1/5,2/5]) */
invert(M).v;                    /* matrix([1/5],[3/5]) */
```

These commands are clear for small symbolic demonstrations. Numerically, explicitly forming A^{-1} is usually less stable and more expensive than solving $Ax = b$. The Marchenko capstone therefore solves linear systems, although its compact helper omits pivoting and is suitable only for the small baseline.

Covariance and contractions

Model 9 constructs

$$C_{\mathrm{EFT}} = \tau\, Y L L^T Y,$$

then normalizes entries to a correlation matrix. Model 12 propagates a covariance as $\Sigma_f = S \Sigma_0 S^T$. These are more than matrix recipes: symmetry, positivity, and dimensional consistency are invariants that the tests can check independently of individual entries.

Chapter 11

Scripts, loading, output, and shell status

Abstract. Maxima source files connect parameter and unit declarations, symbolic representations, numerical methods, explicit reports, regression assertions, and shell exit behavior. This chapter explains the supplied shell runners, file-loading functions, formatted output, data export, and a minimal driver/test pair.

Keywords. Batch Scripts, Batchload, Regression Tests, Shell Status, Data Export

What the shell runners guarantee

The two supplied runners use `set -euo pipefail`, resolve the workspace root from their own location, and start each file in a fresh Maxima process. The glob `[0-9][0-9]_*.mac` preserves the numbered order. A failure stops the suite and becomes a nonzero shell status.

Tests call `batchload` on their model, so the model report appears before the final PASS line. This is expected: the test is executing the real driver, then adding assertions in the same Maxima process.

Load, batch, and batchload

`load("name")` searches Maxima's configured load path and is normally used for installed packages. `batch("file.mac")` executes a file while showing its input and output, which is useful for a teaching transcript. `batchload("file.mac")` executes a file in the current session without that transcript and retains every resulting definition. The paired tests use `batchload` so they can inspect functions, matrices, roots, and parameters created by the corresponding driver.

Relative file names are resolved from Maxima's current working directory, not from the directory containing the calling `.mac` file. The shell runners therefore change to the workspace root before starting Maxima.

Print versus data export

`print` emits Maxima-readable values. `printf` provides field widths and numeric formats such as `~12,6e`. These reports are designed for humans. They are not automatically robust CSV because matrices, lists, spaces, and line wrapping remain Maxima syntax. A machine-readable table has an explicit delimiter, header, precision, and unit convention.

In calls such as `printf(true,"...",value)`, the first argument `true` selects the current standard output. A file stream may be supplied instead, and `with_stdout` can temporarily redirect that standard output to a data file.

The recurring format directives are `~a` for general display, `~d` for an integer, `~w,df` for a width-w fixed-point field with d digits after the decimal, `~w,de` for scientific notation, and `~%` for a newline. One executable example is

```
printf(true,"row=~a i=~2d fixed=~10,4f sci=~12,4e~%",
       [1,2],3,float(%pi),float(%pi))$
```

Scope of a regression result

The supplied regression files evaluate exact identities, limiting cases, floating residuals, and structural invariants. These are properties of the programmed baseline rather than a convergence or conditioning analysis. A passing test says that the programmed baseline remains internally consistent with its chosen identities and stored parameter regime. It does not prove that a grid is converged, a branch is physical, an inverse problem is unique, an approximation is controlled outside its stated regime, or an open problem has been solved.

A minimal driver and paired test

The following `toy_model.mac` defines a function, finds a root, and prints a report.

Listing 11.1: Minimal model driver

```
kill(all)$
display2d:false$

f(x) := x^2 - 2$
root : find_root(f(x),x,1.0,2.0)$
print("root =",root)$
```

The paired `toy_test.mac` batch-loads the model. Because `batchload` retains the driver's definitions, the test can evaluate `f(root)`.

Listing 11.2: Minimal paired regression test

```
batchload("toy_model.mac")$
residual : abs(float(f(root)))$
if not is(residual < 1.0e-12) then
  error("root residual",residual)$
print("TEST OK")$
```

From the directory containing both files, the shell command is

```
maxima --very-quiet --no-init --suppress-input-echo --quit-on-error \
  --batch=toy_test.mac
```

Success prints the model report and `TEST OK`, then returns shell status zero. A failed assertion calls `error`, prints the residual that caused the failure, and returns a nonzero status. This is exactly the control flow of the larger paired files.

Part II

Scalar models and exact identities

Chapter 12

Finite-coupling decoherence and the Zeno layer

Abstract. The first full case study combines delayed scalar functions, exact differentiation, a Laplace transform, nested parameter loops, and a strong-coupling inner limit. It shows how the driver report and paired test divide physical output from algebraic verification.

Keywords. Quantum Zeno Effect, Decoherence, Delayed Functions, Laplace Transform, Asymptotic Scaling

Broader background on Zeno dynamics, anti-Zeno behavior, and non-Markovianity is provided by Refs. [1–3].

The physical question

The first complete model asks how a coherence changes when a dephasing coupling λ is finite but can become large. Two contributions are added to a dephasing exponent:

$$G_{\mathrm{Oh}}(t) = \frac{\eta}{2}\log(1+\omega_c^2 t^2), \tag{12.1}$$

$$G_{\mathrm{str}}(t) = \sigma^2 t_c^2\left(e^{-t/t_c} + \frac{t}{t_c} - 1\right), \tag{12.2}$$

$$C_\lambda(t) = \exp[-\lambda^2(G_{\mathrm{Oh}} + G_{\mathrm{str}})]. \tag{12.3}$$

The instantaneous rate is $\gamma_\lambda(t) = \lambda^2 G'(t)$. The structured correlation $C_{\mathrm{str}}(t) = \sigma^2 e^{-t/t_c}$ has Laplace transform

$$\widetilde{C}_{\mathrm{str}}(z) = \frac{\sigma^2}{z + 1/t_c}.$$

Every formula in this model is scalar, and each printed column has a direct mathematical meaning.

Parameters and scan grids

The exact rational assignments `eta:1/10`, `sigma2:1/20`, and `tc:3/2` make symbolic differentiation and limits clean. Four lists then define two experiments: an outer scan at fixed laboratory time and an inner scan at fixed scaled time $\tau = \lambda t$.

The driver deliberately defines `rate` from the already differentiated closed form. The paired test differentiates `Goh` and `Gstr` independently and verifies that this rate was transcribed correctly.

Deriving the inner Gaussian

At short time,

$$\log(1+\omega_c^2 t^2) = \omega_c^2 t^2 + O(t^4), \tag{12.4}$$

$$e^{-t/t_c} + t/t_c - 1 = \frac{t^2}{2t_c^2} + O(t^3). \tag{12.5}$$

Therefore

$$G(t) = \frac{1}{2}(\eta\omega_c^2 + \sigma^2)t^2 + O(t^3).$$

With $t = \tau/\lambda$,

$$\lambda^2 G(\tau/\lambda) \longrightarrow \frac{1}{2}(\eta\omega_c^2 + \sigma^2)\tau^2,$$

and the limiting coherence is a Gaussian independent of λ. This explains the code rather than merely translating it:

```
short_variance : eta*wc^2+sigma2$
inner_gaussian(tau) := exp(-short_variance*tau^2/2)$
inner_exact(lam,tau) := coherence(lam,tau/lam)$
```

Reading the tables

The outer table shows that increasing λ reduces coherence at the same time. The inner table asks a subtler question: at $\tau = 1$, does the exact finite-coupling value approach the strong-coupling Gaussian? The default run gives

$$C_2(1/2) = 0.7087, \qquad C_{32}(1/32) = 0.4410, \qquad C_{\text{inner}}(1) = 0.4382.$$

The decreasing difference is evidence for the scaling limit at these sample points. The symbolic limit in the test is the stronger algebraic check.

Regression identities

The paired test has four exact residuals:

1. G'_{Oh} minus the Ohmic rate term;
2. G'_{str} minus the structured rate term;
3. the strong-coupling inner exponent minus its Gaussian target;
4. the stored rational kernel minus Maxima's Laplace transform.

It then checks one numerical ordering, $C_4(1/2) < C_2(1/2)$. The exact residuals use `ratsimp` or `radcan` before `equal`; the ordering uses `float` and `is`.

Model scope. The model is an exactly soluble commuting pure-dephasing baseline. A few finite-λ samples do not establish a uniform asymptotic error bound, and the scalar rational kernel is not the projected operator-valued kernel needed for noncommuting Zeno couplings.

Complete driver

Listing 12.1: Finite-coupling dephasing driver (`05_nonmarkovian_zeno.mac`)

```
/* Finite-coupling dephasing with Ohmic and exponential memory. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

/* Editable reservoir parameters (dimensionless units). */
eta:1/10$                       /* Ohmic strength */
wc:4$                           /* Ohmic cutoff */
sigma2:1/20$                    /* C_str(0) */
tc:3/2$                         /* structured-memory time */

Goh(t):=eta*log(1+wc^2*t^2)/2$
```

```
Gstr(t):=sigma2*tc^2*(exp(-t/tc)+t/tc-1)$
Gtot(t):=Goh(t)+Gstr(t)$
coherence(lam,t):=exp(-lam^2*Gtot(t))$
rate(lam,t):=lam^2*(eta*wc^2*t/(1+wc^2*t^2)
                  +sigma2*tc*(1-exp(-t/tc)))$

/* Rational Laplace transform of C_str(t)=sigma2 exp(-t/tc). */
Cstr_tilde(z):=sigma2/(z+1/tc)$
memory_kernel(z,lam):=lam^2*Cstr_tilde(z)$

lambda_grid:[1/2,1,2,4]$
inner_lambda_grid:[2,4,8,16,32]$
time_grid:[1/50,1/10,1/2,1]$
print("outer table: [lambda,t,coherence,instantaneous_rate]")$
for lam in lambda_grid do
  for tv in time_grid do
    print([lam,tv,float(coherence(lam,tv)),float(rate(lam,tv))])$

/* The inner coordinate tau=lambda*t resolves the Zeno layer. */
tau0:1$
short_variance:eta*wc^2+sigma2$
inner_gaussian(tau):=exp(-short_variance*tau^2/2)$
inner_exact(lam,tau):=coherence(lam,tau/lam)$
print("inner table: [lambda,tau,C_exact,C_Gaussian]")$
for lam in inner_lambda_grid do
  print([lam,tau0,float(inner_exact(lam,tau0)),
         float(inner_gaussian(tau0))])$

print("structured Laplace kernel at lambda=2:",
      memory_kernel(z,2))$
```

References

[1] Paolo Facchi and Saverio Pascazio. “Quantum Zeno dynamics: mathematical and physical aspects”. In: *Journal of Physics A: Mathematical and Theoretical* 41 (2008), p. 493001. DOI: 10.1088/1751-8113/41/49/493001. URL: https://arxiv.org/abs/0903.3297.

[2] A. G. Kofman and G. Kurizki. “Acceleration of quantum decay processes by frequent observations”. In: *Nature* 405 (2000), pp. 546–550. DOI: 10.1038/35014537.

[3] Heinz-Peter Breuer, Elsi-Mari Laine, Jyrki Piilo, and Bassano Vacchini. “Non-Markovian dynamics in open quantum systems”. In: *Reviews of Modern Physics* 88 (2016), p. 021002. DOI: 10.1103/RevModPhys.88.021002. URL: https://arxiv.org/abs/1505.01385.

Chapter 13

Exact integration and scaling in orbital-free DFT

Abstract. A normalized Gaussian density provides a compact setting for exact improper integration, radial derivatives, functional energy terms, coordinate scaling, and response matching. The chapter also shows why a symbolic simplification must precede evaluation at a removable radial singularity.

Keywords. Orbital-free DFT, Gaussian Density, Exact Integration, Radial Laplacian, Coordinate Scaling, Linear Response

For the orbital-free kinetic-energy and modern functional context, see Refs. [1, 2].

The Gaussian baseline

For electron number N_e and exponent $a > 0$, the spherical density is

$$n(r) = N_e \left(\frac{a}{\pi}\right)^{3/2} e^{-ar^2}.$$

The normalization is not assumed; the test derives it:

$$4\pi \int_0^\infty r^2 n(r)\, dr = N_e.$$

The model evaluates a Thomas–Fermi plus scaled von Weizsaecker kinetic energy,

$$T_s[n] = C_F \int n^{5/3}\, d^3r + \lambda_W T_W[n], \qquad T_W[n] = \frac{1}{8} \int \frac{|\nabla n|^2}{n}\, d^3r,$$

with $C_F = 3(3\pi^2)^{2/3}/10$.

Why the Euler potential is simplified first

The von Weizsaecker functional derivative can be written

$$v_W(r) = \frac{1}{8}\left[\frac{n'(r)^2}{n(r)^2} - 2\frac{n''(r) + 2n'(r)/r}{n(r)}\right].$$

Direct substitution at $r = 0$ appears to contain $0/0$. After symbolic simplification for the Gaussian it becomes

$$v_W(r) = \frac{3a}{2} - \frac{a^2 r^2}{2},$$

which is regular. The test constructs the derivative form, applies `ratsimp`, and compares it with the compact function `vW`.

Closed energy formulas

The driver stores the analytically integrated results as functions of a trial exponent `aa`:

$$T_{\mathrm{TF}}(a) = C_F N_e^{5/3} \frac{a}{\pi} \left(\frac{3}{5}\right)^{3/2}, \tag{13.1}$$

$$T_W(a) = \frac{3N_e a}{4}. \tag{13.2}$$

Keeping `aa` as a function argument makes the scaling experiment simple. Under $n_\gamma(r) = \gamma^3 n(\gamma r)$, the Gaussian exponent changes as $a \mapsto \gamma^2 a$, hence $T_s[n_\gamma] = \gamma^2 T_s[n]$.

The printed ratios are 1 to floating accuracy for $\gamma = 1/2, 1, 2$. The test proves the same identity symbolically with `ratsimp`.

The linear-response identity

At uniform density n_0, the model kernel is

$$K_{\mathrm{model}}(q) = \frac{10C_F}{9} n_0^{-1/3} + \lambda_W \frac{q^2}{4n_0}.$$

The small-q Lindhard inverse response is

$$K_{\mathrm{L}}(q) = \frac{\pi^2}{k_F} \left(1 + \frac{q^2}{12k_F^2}\right), \qquad k_F = (3\pi^2 n_0)^{1/3}.$$

The baseline value $\lambda_W = 1/9$ makes these agree through q^2. This is why changing `lambdaW` breaks the exact response regression: the test is protecting a deliberate constraint, not merely a stored decimal.

Computed output

The default report separates four ideas:

- a density and Euler-potential profile at four radii;
- T_{TF}, T_W, the weighted gradient term, and total T_s;
- response values at $q/k_F = 0, 1/2, 1$;
- coordinate-scaling ratios.

At the baseline, $T_s \approx 1.21216$ atomic units, and every scaling ratio is 1 up to floating rounding.

The Gaussian density is normalizable only for $a > 0$; negative a also moves its fractional powers onto complex branches.

Model scope. This calculation evaluates a fixed analytic density. It is not a variational orbital-free minimizer, does not test real-space grid convergence or ionic forces, and matches the Lindhard kernel only through the displayed order.

Complete driver

Listing 13.1: Gaussian orbital-free DFT driver (`07_orbital_free_dft.mac`)

```
/* Orbital-free diagnostics for a normalized spherical Gaussian density. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

```

```
/* Atomic units and spin-unpolarized Thomas--Fermi coefficient. */
Ne:2$
a:4/5$
lambdaW:1/9$                    /* second-order gradient expansion */
CF:3*(3*%pi^2)^(2/3)/10$

nrad(r):=Ne*(a/%pi)^(3/2)*exp(-a*r^2)$
vTF(r):=5*CF*nrad(r)^(2/3)/3$
vW(r):=3*a/2-a^2*r^2/2$
vEuler(r):=vTF(r)+lambdaW*vW(r)$

TTF(aa):=CF*Ne^(5/3)*(aa/%pi)*(3/5)^(3/2)$
TW(aa):=3*Ne*aa/4$
Ts(aa):=TTF(aa)+lambdaW*TW(aa)$

print("density table: [r,n(r),delta_Ts/delta_n]")$
for rv in [0,1/2,1,2] do
  print([rv,float(nrad(rv)),float(vEuler(rv))])$

print("energies: [T_TF,T_W,lambdaW*T_W,T_s] =",
      [float(TTF(a)),float(TW(a)),float(lambdaW*TW(a)),float(Ts(a))])$

/* Uniform-gas Hessian versus the Lindhard small-q expansion. */
n0:1/50$
kF:(3*%pi^2*n0)^(1/3)$
Kmodel(q):=10*CF*n0^(-1/3)/9+lambdaW*q^2/(4*n0)$
KLindhard2(q):=%pi^2/kF*(1+q^2/(12*kF^2))$
print("response table: [q/kF,K_model,K_Lindhard_to_q2]")$
for qfrac in [0,1/2,1] do
  print([qfrac,float(Kmodel(qfrac*kF)),
         float(KLindhard2(qfrac*kF))])$

/* Coordinate scaling n_gamma(r)=gamma^3 n(gamma*r) sends a to gamma^2 a. */
print("scaling table: [gamma,T_s[n_gamma],T_s/(gamma^2 T_s[n])]")$
for gam in [1/2,1,2] do
  print([gam,float(Ts(gam^2*a)),
         float(Ts(gam^2*a)/(gam^2*Ts(a)))])$
```

References

[1] Mel Levy and H. Ou-Yang. “Exact properties of the Pauli potential for the square root of the electron density and the kinetic energy functional”. In: *Physical Review A* 38 (1988), pp. 625–629. DOI: 10.1103/PhysRevA.38.625.

[2] Wenfei Mi, Kai Luo, S. B. Trickey, and Michele Pavanello. “Orbital-Free Density Functional Theory: An Attractive Electronic Structure Method for Large-Scale First-Principles Simulations”. In: *Chemical Reviews* 123 (2023), pp. 12039–12104. DOI: 10.1021/acs.chemrev.2c00758.

Chapter 14

Complex scale invariance in the Efimov ladder

Abstract. This case study introduces logarithmic limit cycles, discrete scale invariance, complex energies, width conventions, and controlled effective-range and spin-orbit deformations. Maxima's complex trigonometric functions turn the analytic cycle into an executable level table.

Keywords. Efimov Physics, Discrete Scaling, Limit Cycle, Complex Energy, Effective Range, Spin-orbit Coupling

Reviews of Efimov universality and controlled range corrections include Refs. [1–4].

A limit cycle in logarithmic scale

The running coupling is represented by

$$u(\Lambda) = s_0 \log(\Lambda/\Lambda_*) + i\eta_*, \tag{14.1}$$

$$H(\Lambda) = \frac{\cos[u(\Lambda) + \arctan s_0]}{\cos[u(\Lambda) - \arctan s_0]}. \tag{14.2}$$

Define $\lambda_0 = e^{\pi/s_0}$. Then

$$u(\lambda_0\Lambda) = u(\Lambda) + \pi,$$

and both numerator and denominator change sign, leaving H unchanged. The loop `q:0 thru 4` samples one complete log-cycle in quarter-cycle steps. The first and last printed complex values therefore agree.

The leading and corrected ladders

The leading momentum ladder is

$$\kappa_n^{\mathrm{LO}} = \kappa_* e^{-\pi n/s_0},$$

so successive momenta and energies scale by $e^{-\pi/s_0}$ and $e^{-2\pi/s_0}$. The illustrative correction is

$$\kappa_n^{\mathrm{NLO}} = \kappa_n^{\mathrm{LO}} \left[1 + c_r r_e \kappa_n^{\mathrm{LO}} + c_{\mathrm{so}} \left(\frac{k_{\mathrm{so}}}{\kappa_n^{\mathrm{LO}}} \right)^2 \right].$$

The two corrections behave oppositely down the ladder. The range term shrinks for shallow levels; the spin-orbit term grows because κ_n becomes small. The diagnostic

$$\epsilon_n = |r_e \kappa_n^{\mathrm{LO}}| + (k_{\mathrm{so}}/\kappa_n^{\mathrm{LO}})^2$$

signals when the displayed expansion is no longer controlled.

Complex energy and width convention

The model uses

$$E_n = -(\kappa_n^{\mathrm{NLO}})^2 e^{2i\eta_*/s_0}, \qquad \Gamma_n = -2\,\mathrm{Im}\,E_n.$$

For the chosen sign convention, the imaginary energy is negative and the width is positive. `rectform` is applied before `imagpart`. Arbitrary parameter changes can reverse this sign; the positive-width test protects the baseline convention.

Regression identities

The test uses four distinct strategies:

1. `trigsimp` proves the π-periodicity of the cosine ratio;
2. `radcan` proves the exact LO ladder ratio;
3. `ev(...,re=0,kso=0)` proves that corrections switch off;
4. a floating cycle residual and positive-width inequalities check the numerical baseline.

Notice that the variable `re` means effective range, not "real part." The functions for real and imaginary components are spelled `realpart` and `imagpart`.

Model scope. The range and spin-orbit shifts are a parameterized EFT ansatz. The program does not discretize and renormalize an NLO STM equation, locate poles on a specified sheet, or demonstrate cutoff independence beyond the analytic limit cycle built into the baseline.

Complete driver

Listing 14.1: Complex Efimov limit-cycle driver (`06_efimov_corrections.mac`)

```
/* Complex Efimov limit cycle and an NLO-deformed level ladder. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

/* Editable EFT inputs; hbar^2/m=1. */
s0:1.00624$
LambdaStar:1$
etaStar:0.08$                   /* deep-channel loss phase */
kappaStar:1$
re:0.05$                        /* effective range */
kso:1.0e-5$                     /* infrared SOC momentum */
cr:0.4$
cso:0.2$

u(L):=s0*log(L/LambdaStar)+%i*etaStar$
Hrun(L):=cos(u(L)+atan(s0))/cos(u(L)-atan(s0))$
lambda0:exp(%pi/s0)$

print("cutoff table: [Lambda,Re(H),Im(H)]")$
for q:0 thru 4 do block([Lv,Hv],
  Lv:LambdaStar*exp(q*%pi/(4*s0)),
  Hv:float(rectform(Hrun(Lv))),
  print([float(Lv),realpart(Hv),imagpart(Hv)]))$

kappaLO(n):=kappaStar*exp(-%pi*n/s0)$
range_shift(n):=cr*re*kappaLO(n)$
soc_shift(n):=cso*(kso/kappaLO(n))^2$
kappaNLO(n):=kappaLO(n)*(1+range_shift(n)+soc_shift(n))$
validity(n):=abs(re*kappaLO(n))+(kso/kappaLO(n))^2$
```

```
Elevel(n):=-kappaNLO(n)^2*exp(2*%i*etaStar/s0)$
width(n):=-2*imagpart(rectform(Elevel(n)))$

print("level table: [n,kappa_NLO,Re(E),width,EFT_parameter]")$
for n:0 thru 3 do block([En],
  En:float(rectform(Elevel(n))),
  print([n,float(kappaNLO(n)),realpart(En),
         float(width(n)),float(validity(n))]))$

print("LO discrete factors: momentum =",float(1/lambda0),
      ", energy =",float(1/lambda0^2))$
```

References

[1] Eric Braaten and H.-W. Hammer. "Universality in few-body systems with large scattering length". In: *Physics Reports* 428 (2006), pp. 259–390. DOI: 10.1016/j.physrep.2006.03.001. URL: https://arxiv.org/abs/cond-mat/0410417.

[2] Pascal Naidon and Shimpei Endo. "Efimov physics: a review". In: *Reports on Progress in Physics* 80 (2017), p. 056001. DOI: 10.1088/1361-6633/aa50e8. URL: https://arxiv.org/abs/1610.09805.

[3] Chen Ji, Daniel R. Phillips, and Lucas Platter. "The three-boson system at next-to-leading order in an effective field theory for systems with a large scattering length". In: *Annals of Physics* 327 (2012), pp. 1803–1824. DOI: 10.1016/j.aop.2012.02.001. URL: https://arxiv.org/abs/1106.3837.

[4] Chen Ji, Eric Braaten, Daniel R. Phillips, and Lucas Platter. "Universal relations for range corrections to Efimov features". In: *Physical Review A* 92 (2015), p. 030702. DOI: 10.1103/PhysRevA.92.030702. URL: https://arxiv.org/abs/1506.02334.

Part III

Matrices, poles, fields, and propagation

Chapter 15

Matrix rotations and correlated EFT uncertainty

Abstract. A two-channel rotation demonstrates matrix products, coevolution of states and observables, representation invariance, and formatted output. A second construction builds covariance and correlation matrices from shared omitted-order modes.

Keywords. Matrix Rotation, Operator Coevolution, Effective Field Theory, Covariance Matrix, Correlated Uncertainty

The nuclear-EFT and uncertainty-quantification context is developed in Refs. [1–4].

The two-channel representation

The driver uses a two-dimensional surrogate for coupled S and D channels:

$$H_0 = \begin{pmatrix} -2.2246 & 0.62 \\ 0.62 & 1.15 \end{pmatrix}, \quad \psi_0 = \begin{pmatrix} \sqrt{0.94} \\ \sqrt{0.06} \end{pmatrix}, \quad J_0 = \begin{pmatrix} 1 & 0.18 \\ 0.18 & -0.35 \end{pmatrix}.$$

A scale-dependent rotation

$$U(s) = \begin{pmatrix} \cos\theta(s) & -\sin\theta(s) \\ \sin\theta(s) & \cos\theta(s) \end{pmatrix}$$

transforms all three objects:

$$H(s) = UH_0U^T, \qquad \psi(s) = U\psi_0, \qquad J(s) = UJ_0U^T.$$

The prescribed angle approaches a value that suppresses the off-diagonal Hamiltonian element.

Why the observable must coevolve

The expectation helper is

```
expect(p,O) := transpose(p).O.p$
```

Because $U^TU = I$,

$$\psi(s)^T J(s)\psi(s) = \psi_0^T U^T (UJ_0U^T)U\psi_0 \tag{15.1}$$

$$= \psi_0^T J_0 \psi_0. \tag{15.2}$$

The coevolved column in the report remains approximately 1.00449526. The column computed with the unevolved J_0 drifts as s changes. Maxima is not merely multiplying matrices here; the invariant distinguishes a consistent representation change from an inconsistent one.

`transpose` is adequate because every baseline entry is real. For a complex state or operator, a Hermitian expectation requires conjugation as well as transposition.

A covariance from shared omitted-order modes

Three observables are coupled to two shared latent modes through a 3×2 loading matrix L. The reference scales form a diagonal 3×3 matrix Y. With expansion parameter Q and completed order k, the geometric tail is

$$\tau = \frac{Q^{2(k+1)}}{1 - Q^2}.$$

The model covariance is

$$C_{\mathrm{EFT}} = \tau Y L L^T Y, \qquad C_{\mathrm{tot}} = C_{\mathrm{EFT}} + C_{\mathrm{num}}.$$

Finally, `genmatrix(lambda([i,j],...))` constructs

$$\rho_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} C_{jj}}}.$$

The output contains substantial positive correlations, such as $\rho_{12} \approx 0.881$. Those off-diagonal values arise from the shared columns of L, not from a post-processing choice.

Formatted output

The directive

```
printf(true,"~5,2f  ~10,6f  ~12,8f~%",x,y,z)$
```

uses fixed-width floating fields followed by a newline `~%`. This makes a human-readable table. The command `matrixmap('float,Ctot)` applies floating conversion entry by entry while keeping matrix structure.

Regression conditions

For every printed flow scale, the test checks coevolution invariance to 10^{-10}. It also checks that the off-diagonal Hamiltonian magnitude is smaller at $s = 4$ than at $s = 0$, that an off-diagonal covariance is nonzero, and that $\det C_{\mathrm{tot}} > 0$.

Model scope. The angle is prescribed; the program does not integrate an SRG flow equation. The covariance is a small latent-mode example, not a calibrated probability law for omitted interaction and current operators. Positivity at the baseline does not prove positivity for arbitrary edited loadings.

Complete driver

Listing 15.1: Two-channel flow and covariance driver (`09_tensor_force_eft.mac`)

```
/* Two-channel S--D similarity flow and correlated EFT predictions. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

H0 : matrix([-2.2246,0.62],[0.62,1.15])$  /* MeV */
psi0 : matrix([sqrt(0.94)],[sqrt(0.06)])$
J0 : matrix([1.00,0.18],[0.18,-0.35])$
theta_inf : 0.5*atan(2*H0[1,2]/(H0[2,2]-H0[1,1]))$
theta(s) := theta_inf*(1-exp(-s/0.8))$
U(s) := matrix([cos(theta(s)),-sin(theta(s))],
               [sin(theta(s)), cos(theta(s))])$
H(s) := U(s).H0.transpose(U(s))$
psi(s) := U(s).psi0$
Jflow(s) := U(s).J0.transpose(U(s))$
```

```
expect(p,O) := transpose(p).O.p$

print("SRG-like two-channel flow (energies in MeV)")$
print(" s       H_SD       <J> coevolved    <J0> inconsistent")$
for sv in [0,0.25,0.5,1,2,4] do block([hs:H(sv),ps:psi(sv)],
  printf(true,"~5,2f  ~10,6f    ~12,8f      ~12,8f~%",
    float(sv),float(hs[1,2]),float(expect(ps,Jflow(sv))),
    float(expect(ps,J0))))$

/* Shared omitted-order modes correlate three tensor observables. */
Q : 0.28$
kdone : 3$
L : matrix([1.00,0.25],[0.72,-0.18],[0.42,0.55])$
yref : matrix([1.00],[0.31],[0.86])$
tail : Q^(2*(kdone+1))/(1-Q^2)$
Y : matrix([yref[1,1],0,0],[0,yref[2,1],0],[0,0,yref[3,1]])$
Ceft : tail*Y.L.transpose(L).Y$
Cnum : matrix([2e-8,0,0],[0,4e-9,0],[0,0,1e-8])$
Ctot : Ceft+Cnum$
Corr : genmatrix(lambda([i,j],Ctot[i,j]/sqrt(Ctot[i,i]*Ctot[j,j])),3,3)$
print("Correlated truncation covariance for [G_C,T_20,G_M]")$
print(matrixmap('float,Ctot))$
print("Corresponding correlation matrix")$
print(matrixmap('float,Corr))$
```

References

[1] R. Machleidt and D. R. Entem. “Chiral effective field theory and nuclear forces”. In: *Physics Reports* 503 (2011), pp. 1–75. DOI: 10.1016/j.physrep.2011.02.001. URL: https://arxiv.org/abs/1105.2919.

[2] H.-W. Hammer, Sebastian König, and U. van Kolck. “Nuclear effective field theory: status and perspectives”. In: *Reviews of Modern Physics* 92 (2020), p. 025004. DOI: 10.1103/RevModPhys.92.025004. URL: https://arxiv.org/abs/1906.12122.

[3] R. J. Furnstahl, D. R. Phillips, and S. Wesolowski. “A recipe for EFT uncertainty quantification in nuclear physics”. In: *Journal of Physics G: Nuclear and Particle Physics* 42 (2015), p. 034028. DOI: 10.1088/0954-3899/42/3/034028. URL: https://arxiv.org/abs/1407.0657.

[4] J. A. Melendez, R. J. Furnstahl, D. R. Phillips, M. T. Pratola, and S. Wesolowski. “Quantifying correlated truncation errors in effective field theory”. In: *Physical Review C* 100 (2019), p. 044001. DOI: 10.1103/PhysRevC.100.044001. URL: https://arxiv.org/abs/1904.10581.

Chapter 16

Scattering poles and implicit sensitivities

Abstract. A small inverse-amplitude determinant becomes a transparent laboratory for polynomial extraction, analytic roots, the implicit function theorem, finite-difference verification, conditioning, and local root isolation.

Keywords. Scattering Poles, Determinant, Implicit Differentiation, Root Sensitivity, Conditioning, Residuals

For nonlinear spectral problems and modern root-certification context, see Refs. [1, 2].

From a matrix to a scalar pole equation

The inverse-amplitude surrogate is

$$A(E,g) = \begin{pmatrix} E - e_1 & -g(1+\beta E) \\ -g(1+\beta E) & E - e_2 \end{pmatrix}.$$

Its poles are zeros of

$$D(E,g) = \det A = (E-e_1)(E-e_2) - g^2(1+\beta E)^2.$$

After `expand`, `coeff(D(E,g),E,n)` extracts the quadratic, linear, and constant coefficients. The function `pole(branch,g)` applies the quadratic formula with `branch` equal to -1 or $+1$.

The decimal parameters mean this determinant is approximate from the start. It is a numerical baseline rather than an exact polynomial identity over rational coefficients.

Implicit differentiation

Along a simple pole branch, $D(E_p(g), g) = 0$. Differentiation gives

$$D_E \frac{dE_p}{dg} + D_g = 0, \qquad \frac{dE_p}{dg} = -\frac{D_g}{D_E}.$$

The driver computes `DEexpr` and `Dgexpr` once, then substitutes the pole and coupling. A centered finite difference of the explicit pole formula serves as an independent implementation. At the baseline, analytic and finite-difference sensitivities agree to the displayed digits.

Residual, conditioning, and isolation

Three quantities answer different questions:

Residual $|D(E_p)|$
: Is the computed number close to a zero of the programmed determinant?

Condition factor $1/|D_E(E_p)|$
: How strongly can a determinant perturbation move a simple pole?

Diagnostic margin

For a quadratic and disk radius r, the driver evaluates

$$|D_E(E_p)|r - |a|r^2 - |D(E_p)|.$$

A positive value is a floating local isolation check based on the linear term dominating the quadratic remainder and residual.

The test also requires the two radius-r disks not to overlap. Both baseline residuals are near machine zero, and the pole separation is about 1.775, far larger than $2r = 0.16$.

Degenerate and coalescing roots

If the quadratic coefficient `aa(g)` vanishes, the quadratic formula is the wrong representation and the equation is linear. If the discriminant or D_E approaches zero, two simple poles coalesce: sensitivity becomes large, branch labels become unstable, and a cluster derivative is needed.

Model scope. The Rouché-style margin is evaluated in ordinary floating arithmetic; it is not a certified interval bound. The model assumes two simple roots of a quadratic surrogate. Continuum branch cuts, multiple poles, discretization error, and globally consistent branch continuation are outside this driver.

Complete driver

Listing 16.1: Differentiable two-channel pole driver (`10_differentiable_scattering.mac`)

```
/* Energy-dependent rank-two inverse amplitude: poles and sensitivities. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

e1 : -1.20$ e2 : 0.45$ beta : 0.25$ g0 : 0.35$
A(E,g) := matrix([E-e1,-g*(1+beta*E)],
                 [-g*(1+beta*E),E-e2])$
D(E,g) := expand(determinant(A(E,g)))$
aa(g) := coeff(D(E,g),E,2)$
bb(g) := coeff(D(E,g),E,1)$
cc(g) := coeff(D(E,g),E,0)$
pole(branch,g) := (-bb(g)+branch*sqrt(bb(g)^2-4*aa(g)*cc(g)))/(2*aa(g))$
DEexpr : diff(D(E,g),E)$
Dgexpr : diff(D(E,g),g)$
dEdg(Ep,gv) := -subst([E=Ep,g=gv],Dgexpr)/subst([E=Ep,g=gv],DEexpr)$

hfd : 1e-5$
radius : 0.08$
print("Energy-dependent two-channel pole map")$
print("branch       E_pole       dE/dg analytic    dE/dg finite-diff")$
for branch in [-1,1] do block([Ep,sa,sf],
  Ep:pole(branch,g0),
  sa:dEdg(Ep,g0),
  sf:(pole(branch,g0+hfd)-pole(branch,g0-hfd))/(2*hfd),
  printf(true," ~2d      ~12,8f      ~12,8f        ~12,8f~%",
    branch,Ep,sa,sf))$

/* Floating Rouche diagnostic; rigorous certification needs interval bounds. */
print("Floating pole-isolation diagnostic for radius",radius)$
print("branch     residual        condition 1/|D_E|    diagnostic margin")$
for branch in [-1,1] do block([Ep,res,cond,diag_margin],
  Ep:pole(branch,g0),
  res:abs(D(Ep,g0)),
```

```
  cond:1/abs(subst(Ep,E,diff(D(E,g0),E))),
  diag_margin:abs(subst(Ep,E,diff(D(E,g0),E)))*radius
              -abs(aa(g0))*radius^2-res,
  printf(true," ~2d      ~12,4e       ~12,6f        ~12,6e~%",
    branch,res,cond,diag_margin))$
print("Pole separation =",float(abs(pole(1,g0)-pole(-1,g0))))$
```

References

[1] Wolf-Jürgen Beyn. "An integral method for solving nonlinear eigenvalue problems". In: *Linear Algebra and its Applications* 436 (2012), pp. 3839–3863. DOI: 10.1016/j.laa.2011.03.030.

[2] Jonathan Ben-Artzi, Marco Marletta, and Frank Rösler. "Computing scattering resonances". In: *Journal of the European Mathematical Society* (2023). DOI: 10.4171/JEMS/1258. URL: https://arxiv.org/abs/2006.03368.

Chapter 17

A deformed supercritical Dirac resonance ladder

Abstract. The supercritical Dirac model combines a geometric momentum ladder with screening, lattice, valley, and loss deformations. It develops complex dispersion, resonance widths, switch-off tests, and the limits of an algebraically constructed matching determinant.

Keywords. Dirac Resonances, Supercritical Coupling, Complex Dispersion, Valley Splitting, Resonance Width

The supercritical Dirac-Coulomb setting and atomic-collapse observations are reviewed in Refs. [1–3].

The supercritical scale

For coupling g and angular parameter j, the baseline assumes $g > |j|$ and defines

$$\beta = \sqrt{g^2 - j^2}, \qquad k_n^{(0)} = k_{\mathrm{UV}} \exp\left[-\frac{\phi_{\mathrm{core}} + \pi n}{\beta}\right].$$

The ideal consecutive momentum ratio is $e^{-\pi/\beta}$, about 0.05614 for the default parameters. If $g \le |j|$, β is no longer the real supercritical exponent assumed by this construction.

Four distinct deformations

The code keeps physical mechanisms separate before combining them:

$$\delta_{\mathrm{scr}}(k) = -\frac{c_{\mathrm{scr}}}{1 + (k\ell_{\mathrm{scr}})^2}, \tag{17.1}$$

$$\delta_{\mathrm{lat}}(k) = c_{\mathrm{lat}}(kr_{\mathrm{core}})^2, \tag{17.2}$$

$$\delta_{\mathrm{val}}(k, \tau) = \tau c_{\mathrm{iv}} \frac{e^{-2kr_{\mathrm{core}}}}{1 + (k\ell_{\mathrm{scr}})^2}, \tag{17.3}$$

$$k_{n\tau} = k_n^{(0)}(1 + \delta_{\mathrm{scr}} + \delta_{\mathrm{lat}} + \delta_{\mathrm{val}})e^{i\theta_{\mathrm{loss}}}. \tag{17.4}$$

Here $\tau = \pm 1$ labels the two valleys. Energies are in eV, lengths in nm, momenta in nm^{-1}, and $\hbar v_F$ in eV nm.

Dispersion, widths, and splitting

The corrected momentum is mapped through the negative branch of a gapped Dirac dispersion:

$$E_{n\tau} = -\sqrt{\Delta^2 + (\hbar v_F k_{n\tau})^2}, \qquad \Gamma_{n\tau} = -2\,\mathrm{Im}\, E_{n\tau}.$$

The use of `rectform` before component extraction is important. Both the momentum phase and the outer square root carry branch information.

The default report shows the ladder rapidly approaching the band edge $-\Delta = -0.1$ eV. Widths and valley splittings shrink with level number.

The matching determinant

Define the mean and half-splitting

$$\bar{E}_n = \frac{E_{n+} + E_{n-}}{2}, \qquad V_n = \frac{E_{n+} - E_{n-}}{2}.$$

Then

$$D_{\mathrm{match}}(E, n) = (E - \bar{E}_n)^2 - V_n^2$$

has roots E_{n+} and E_{n-}. Equivalently, it is the determinant of the effective valley matrix with diagonal $\bar{E}_n$ and off-diagonal V_n, up to the chosen basis convention.

The test proves both roots algebraically and restores degeneracy by setting `cIntervalley` to zero with `ev`. Because the determinant is constructed from the same levels, its zero is an algebraic consistency check, not independent microscopic radial matching.

Model scope. The correction functions are a parameterized matching ansatz. The script does not derive them from a radial Dirac or lattice-core Green function, perform Riemann-sheet continuation, or include experimental resolution. Positive widths are tied to the baseline sign convention and parameter range.

Complete driver

Listing 17.1: Supercritical Dirac ladder driver (`08_supercritical_dirac.mac`)

```
/* Screened, lossy, valley-split supercritical Dirac ladder. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

/* Energies in eV and lengths in nm. */
g:6/5$
j:1/2$
beta:sqrt(g^2-j^2)$
kUV:7/10$
phiCore:3/20$
Delta:1/10$
hbarv:329/500$
lscr:5$
rcore:1/5$
cScreen:1/20$
cLattice:1/50$
cIntervalley:1/100$
thetaLoss:1/50$

kbase(n):=kUV*exp(-(phiCore+%pi*n)/beta)$
dScreen(k):=-cScreen/(1+(k*lscr)^2)$
dLattice(k):=cLattice*(k*rcore)^2$
dValley(k,tau):=tau*cIntervalley*exp(-2*k*rcore)
                        /(1+(k*lscr)^2)$
kcorr(n,tau):=kbase(n)*(1+dScreen(kbase(n))
                    +dLattice(kbase(n))+dValley(kbase(n),tau))
                    *exp(%i*thetaLoss)$

Elevel(n,tau):=-sqrt(Delta^2+(hbarv*kcorr(n,tau))^2)$
width(n,tau):=-2*imagpart(rectform(Elevel(n,tau)))$
valley_split(n):=realpart(rectform(Elevel(n,1)-Elevel(n,-1)))$
Emid(n):=(Elevel(n,1)+Elevel(n,-1))/2$
Vintervalley(n):=(Elevel(n,1)-Elevel(n,-1))/2$
Dmatch(E,n):=(E-Emid(n))^2-Vintervalley(n)^2$

```

```
print("ideal momentum ratio =",float(exp(-%pi/beta)))$
print("pole table: [n,Re(E+),Re(E-),width+,width-,valley_split]")$
for n:0 thru 3 do block([Ep,Em],
  Ep:float(rectform(Elevel(n,1))),
  Em:float(rectform(Elevel(n,-1))),
  print([n,realpart(Ep),realpart(Em),
         float(width(n,1)),float(width(n,-1)),
         float(valley_split(n))]))$

print("Dmatch=(E-Emid)^2-Vintervalley^2; n=1 [Emid,Vintervalley] =")$
print([float(rectform(Emid(1))),float(rectform(Vintervalley(1)))])$
```

References

[1] Andrei V. Shytov, Mikhail I. Katsnelson, and Leonid S. Levitov. "Atomic collapse and quasi-Rydberg states in graphene". In: *Physical Review Letters* 99 (2007), p. 246802. DOI: 10.1103/PhysRevLett.99.246802.

[2] Y. Wang, D. Wong, A. V. Shytov, V. W. Brar, S. Choi, Q. Wu, H.-Z. Tsai, W. Regan, A. Zettl, R. K. Kawakami, S. G. Louie, L. S. Levitov, and M. F. Crommie. "Observing atomic collapse resonances in artificial nuclei on graphene". In: *Science* 340 (2013), pp. 734–737. DOI: 10.1126/science.1234320.

[3] E. V. Gorbar, V. P. Gusynin, and O. O. Sobol. "Electron states in the field of charged impurities in two-dimensional Dirac systems". In: *Low Temperature Physics* 44 (2018), pp. 371–400. DOI: 10.1063/1.5034149. URL: https://arxiv.org/abs/1806.05568.

Chapter 18

Structured light as complex vector programming

Abstract. A four-ray angular spectrum turns complex polarization, vector products, nested sums, analytic gradients, and multipole contractions into a readable Maxima program. The regression checks transversality, normalization, and safe sampling away from amplitude nodes.

Keywords. Structured Light, Complex Polarization, Angular Spectrum, Field Gradient, Multipole Amplitudes

The structured-light and twisted-photon setting is developed in Refs. [1–3].

A four-ray angular spectrum

The model samples four azimuths $\phi \in \{0, \pi/2, \pi, 3\pi/2\}$. For each ray it constructs

$$\mathbf{k}(\phi) = (k_t \cos\phi, k_t \sin\phi, k_z)^T, \tag{18.1}$$

$$\mathbf{s}(\phi) = (-\sin\phi, \cos\phi, 0)^T, \tag{18.2}$$

$$\mathbf{p}(\phi) = (k_z \cos\phi/k_0, k_z \sin\phi/k_0, -k_t/k_0)^T. \tag{18.3}$$

The helicity polarization is

$$\boldsymbol{\epsilon}(\phi) = \frac{\mathbf{s}(\phi) + i\,h\mathbf{p}(\phi)}{\sqrt{2}},$$

and the quasi-OAM weight is $e^{i\ell\phi}/4$.

The function `dot3` is intentionally bilinear: it does not conjugate its arguments. This is correct for $\mathbf{k}\cdot\boldsymbol{\epsilon} = 0$ and for transition contractions. The normalization test explicitly supplies `conjugate(ep)` when a Hermitian inner product is required.

Fields and gradients

Each plane-wave phase is

$$e^{i\mathbf{k}\cdot\mathbf{r}}.$$

The electric field sums weighted polarizations. The magnetic field uses the plane-wave relation

$$\mathbf{B} = \frac{\mathbf{k}\times\boldsymbol{\epsilon}}{ck_0} e^{i\mathbf{k}\cdot\mathbf{r}}.$$

Differentiating the phase with respect to coordinate x_i produces ik_i, so `gradE(i,j,...)` represents $\partial_i E_j$. This index order matters in the quadrupole contraction.

Multipole amplitudes

With dipole $\mathbf{d}$, magnetic dipole $\boldsymbol{\mu}$, and quadrupole tensor Q_{ij}, the demonstration evaluates

$$A_{E1} = -\mathbf{d} \cdot \mathbf{E}, \tag{18.4}$$

$$A_{E2} = -\frac{1}{6} \sum_{ij} Q_{ij} \partial_i E_j, \tag{18.5}$$

$$A_{M1} = -\boldsymbol{\mu} \cdot \mathbf{B}. \tag{18.6}$$

The quadrupole is scaled as dipole times the target length a, and $ka \approx 0.1257 < 1$ records the controlled small-target regime.

Four rays possess only discrete C_4 rotational structure; they are not a converged continuous-OAM beam. The report calls this "quasi-OAM" deliberately.

Ratios and nodes

The printed observables are $|E2/E1|$ and $|M1/E1|$. These ratios become singular at an E1 node even when the numerator is finite. The regression test does not prove global regularity; it only verifies that the three chosen sample points have a nonzero E1 denominator.

The remaining checks verify $\mathbf{k} \cdot \boldsymbol{\epsilon} = 0$, magnetic transversality, and complex polarization normalization for all four rays.

Model scope. Fields and target moments use relative units, the angular spectrum has only four rays, and the calculation is monochromatic. It does not supply a pulse-normalized continuum yield, gauge consistency across truncated target bases, or a convergence study in angular sampling.

Complete driver

Listing 18.1: Four-ray structured-light driver (`11_structured_light.mac`)

```
/* Four-ray nonparaxial angular spectrum and E1/E2/M1 diagnostics. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

c : 1.0$ lambda0 : 1.0$ k0 : 2*%pi/lambda0$ /* relative units */
cone : 0.55$ kt : cone*k0$ kz : sqrt(k0^2-kt^2)$
ell_oam : 1$ helicity : 1$
phis : [0,%pi/2,%pi,3*%pi/2]$
dot3(a,b) := sum(a[i,1]*b[i,1],i,1,3)$
cross3(a,b) := matrix([a[2,1]*b[3,1]-a[3,1]*b[2,1]],
 [a[3,1]*b[1,1]-a[1,1]*b[3,1]],
 [a[1,1]*b[2,1]-a[2,1]*b[1,1]])$
kvec(ph) := matrix([kt*cos(ph)],[kt*sin(ph)],[kz])$
svec(ph) := matrix([-sin(ph)],[cos(ph)],[0])$
pvec(ph) := matrix([kz*cos(ph)/k0],[kz*sin(ph)/k0],[-kt/k0])$
eps(ph) := (svec(ph)+%i*helicity*pvec(ph))/sqrt(2)$
weight(ph) := exp(%i*ell_oam*ph)/4$
phase(ph,x,y,z) := exp(%i*dot3(kvec(ph),matrix([x],[y],[z])))$
Efield(x,y,z) := sum(weight(phis[q])*eps(phis[q])
                        *phase(phis[q],x,y,z),q,1,4)$
Bfield(x,y,z) := sum(weight(phis[q])*cross3(kvec(phis[q]),eps(phis[q]))
                        *phase(phis[q],x,y,z)/(c*k0),q,1,4)$
gradE(i,j,x,y,z) := sum(%i*kvec(phis[q])[i,1]*weight(phis[q])
                        *eps(phis[q])[j,1]*phase(phis[q],x,y,z),q,1,4)$

d_scale : 1.0$ a_target : 0.02*lambda0$ ka : k0*a_target$
d : d_scale*matrix([1.0],[0.30],[0.15])$
mu : matrix([0.02],[0.01],[0.03])$
```

```
Qt : d_scale*a_target
     *matrix([0.35,0.08,0.00],[0.08,-0.20,0.05],[0.00,0.05,-0.15])$
AE1(x,y,z) := -dot3(d,Efield(x,y,z))$
AE2(x,y,z) := -sum(sum(Qt[i,j]*gradE(i,j,x,y,z),j,1,3),i,1,3)/6$
AM1(x,y,z) := -dot3(mu,Bfield(x,y,z))$
samples : [[0.08,0.03,0],[0.20,-0.05,0.10],[0.35,0.12,0.20]]$
print("Lengths use lambda0; c=1; E,B,d,mu are relative and Q has d*length units")$
printf(true,"target size a = ~8,5f; ka = ~8,5f; four rays give C4 quasi-OAM~%",
       a_target,float(ka))$
print("Nonparaxial four-ray field and multipole diagnostics")$
print(" (x,y,z)             Re(E_x)     Im(E_x)     |E2/E1|    |M1/E1|")$
for r in samples do block([ef:Efield(r[1],r[2],r[3]),a1,a2,am],
  a1:AE1(r[1],r[2],r[3]), a2:AE2(r[1],r[2],r[3]),
  am:AM1(r[1],r[2],r[3]),
  printf(true,"~a  ~10,6f  ~10,6f  ~10,6f  ~10,6f~%",r,
    realpart(float(ef[1,1])),imagpart(float(ef[1,1])),
    float(abs(a2/a1)),float(abs(am/a1))))$
Elast : Efield(0.35,0.12,0.20)$
Blast : Bfield(0.35,0.12,0.20)$
print("Last sample components: j, Re(E_j), Im(E_j), Re(B_j), Im(B_j)")$
for j thru 3 do printf(true,"~2d  ~11,6f ~11,6f  ~11,6f ~11,6f~%",j,
  realpart(float(Elast[j,1])),imagpart(float(Elast[j,1])),
  realpart(float(Blast[j,1])),imagpart(float(Blast[j,1])))$
```

References

[1] Andrei Afanasev, Carl E. Carlson, and Asmita Mukherjee. *Off-axis excitation of hydrogenlike atoms by twisted photons*. 2013. URL: https://arxiv.org/abs/1305.3650 (visited on 07/16/2026).

[2] Andrei Afanasev, Carl E. Carlson, and Asmita Mukherjee. "High-multipole excitations of hydrogen-like atoms by twisted photons near phase singularity". In: *Journal of Optics* 18 (2016), p. 074013. DOI: 10.1088/2040-8978/18/7/074013. URL: https://arxiv.org/abs/1604.05623.

[3] Christian T. Schmiegelow, Jonas Schulz, Hendrik Kaufmann, Thomas Ruster, Ulrich G. Poschinger, and Ferdinand Schmidt-Kaler. "Transfer of optical orbital angular momentum to a bound electron". In: *Nature Communications* 7 (2016), p. 12998. DOI: 10.1038/ncomms12998.

Chapter 19

Affine symplectic quantum dynamics

Abstract. This advanced propagation model uses hypergeometric entire functions to cross a signed gradient smoothly, composes affine symplectic maps and momentum kicks, retains gravity symbolically for differentiation, and propagates Gaussian covariance to an interferometric contrast.

Keywords. Quantum Dynamics, Symplectic Map, Affine Propagation, Hypergeometric Function, Covariance, Atom Interferometry

Background on light-pulse atom interferometry, relativistic phases, and gradient compensation is available in Refs. [1–3].

Start from the zero-gradient map

Use canonical phase space $x = (z, p)^T$. With constant downward acceleration g and zero gradient, a segment of duration t is

$$S_0(t) = \begin{pmatrix} 1 & t/m \\ 0 & 1 \end{pmatrix}, \qquad d_0(t) = \begin{pmatrix} -gt^2/2 \\ -mgt \end{pmatrix}, \qquad x(t) = S_0 x(0) + d_0.$$

The paired test proves that the general functions reduce exactly to these expressions at $G = 0$.

Entire functions for signed gradient

For positive (G), familiar forms are

$$C(t, G) = \cos(\sqrt{G}t), \quad S(t, G) = \frac{\sin(\sqrt{G}t)}{\sqrt{G}}, \quad K(t, G) = \frac{1 - \cos(\sqrt{G}t)}{G}.$$

The quotients appear singular at (G=0), and a direct square root changes character for negative (G). The driver instead uses their entire series:

$$C = {}_0F_1(;1/2;-Gt^2/4), \tag{19.1}$$

$$S = t\,{}_0F_1(;3/2;-Gt^2/4), \tag{19.2}$$

$$K = \frac{t^2}{2}\,{}_1F_2(1;3/2,2;-Gt^2/4). \tag{19.3}$$

This single representation is finite at zero and analytic for either sign.

The segment map is

$$S_G = \begin{pmatrix} C & S/m \\ -mGS & C \end{pmatrix}, \qquad d_G = \begin{pmatrix} -gK \\ -mgS \end{pmatrix}.$$

Why `grav` remains unassigned

The physical number is stored as `g0`, but the map is built with the free symbol `grav`. The program first obtains a symbolic phase, then computes

```
phase_value : subst(g0,grav,phase)$
scale_g : diff(phase,grav)$
```

Assigning `grav:g0` before the differentiation would erase the symbolic dependence and destroy the scale-factor calculation. This is a concrete use of Maxima's symbolic evaluation model inside an otherwise numerical program.

Tracing the two arms

A momentum kick is an affine translation $x \mapsto x + (0, \Delta p)^T$. At time zero, arm A receives one recoil while arm B does not. At (T), opposite mirror kicks are applied; both arms then propagate to (2T). The final momentum closure includes the output recoil by convention.

The phase contains a large gravity contribution. The program reports both the full phase and

$$\delta\Phi_\Gamma = \Phi + k_{\mathrm{eff}} g_0 T^2,$$

which subtracts the leading uniform-gravity term. At the baseline, numbers of order 10^6 radians cancel to leave a shift of order 10^{-1} radians. The subtraction is therefore sensitive to finite precision.

Covariance and contrast

The initial minimum-uncertainty covariance is

$$\Sigma_0 = \begin{pmatrix} \sigma_z^2 & 0 \\ 0 & \hbar^2/(4\sigma_z^2) \end{pmatrix},$$

and $\Sigma_f = S(2T)\Sigma_0 S(2T)^T$. With symplectic form $J = \left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$ and arm mismatch δ, the contrast model is

$$\mathcal{C} = \exp\left[-\frac{\delta^T J \Sigma_f J^T \delta}{2\hbar^2}\right].$$

The test expands this quadratic form independently component by component.

Regression identities

The paired test checks:

1. symplecticity $S^T J S = J$ for positive, zero, and negative gradients;
2. the exact free-fall limit at zero gradient;
3. composition of two affine segments;
4. agreement of matrix and component forms of the overlap exponent;
5. covariance symmetry and positivity;
6. the physical interval $0 < \mathcal{C} \leq 1$.

Model scope. The model is one-dimensional, quadratic, and uses instantaneous kicks. The contrast assumes a pure Gaussian with the stated covariance. It omits finite pulses, rotations, relativistic clock terms, and correlated noise. SI-scale entries and dominant-phase subtraction are sensitive to numerical precision.

Complete driver

Listing 19.1: Affine symplectic interferometer driver (`12_accelerated_quantum_dynamics.mac`)

```
/* Affine symplectic Mach--Zehnder model with signed gravity gradient. */
kill(all)$
display2d:false$
ratprint:false$
linel:200$

m : 1.44316060e-25$ hbar : 1.054571817e-34$
keff : 8.055e6$ recoil : hbar*keff$
T : 0.20$ Gamma : 3.0e-6$ g0 : 9.81$
z0 : 0.0$ v0 : 0.05$ sigma_z : 1.0e-4$
J : matrix([0,1],[-1,0])$

/* Entire sinc/cosc continuations: exact at G=0 and valid for signed G. */
Cfun(t,G) := hypergeometric([],[1/2],-G*t^2/4)$
Sfun(t,G) := t*hypergeometric([],[3/2],-G*t^2/4)$
Kfun(t,G) := t^2*hypergeometric([1],[3/2,2],-G*t^2/4)/2$
Sseg(t,G) := matrix([Cfun(t,G),Sfun(t,G)/m],
                    [-m*G*Sfun(t,G),Cfun(t,G)])$
dseg(t,grav,G) := matrix([-grav*Kfun(t,G)],[-m*grav*Sfun(t,G)])$
prop(x,t,grav,G) := Sseg(t,G).x+dseg(t,grav,G)$
kick(x,dp) := x+matrix([0],[dp])$
xinit : matrix([z0],[m*v0])$

xA0 : kick(xinit,recoil)$ xB0 : xinit$
xAT : prop(xA0,T,grav,Gamma)$ xBT : prop(xB0,T,grav,Gamma)$
xA2 : prop(kick(xAT,-recoil),T,grav,Gamma)$
xB2 : prop(kick(xBT, recoil),T,grav,Gamma)$
zbar2 : (xA2[1,1]+xB2[1,1])/2$
phase : keff*(z0-xAT[1,1]-xBT[1,1]+zbar2)$
phase_value : subst(g0,grav,phase)$ scale_g : diff(phase,grav)$
gradient_shift : phase_value+keff*g0*T^2$
delta : matrix([xA2[1,1]-xB2[1,1]],
               [xA2[2,1]-xB2[2,1]+recoil])$
S2 : Sseg(T,Gamma).Sseg(T,Gamma)$
d2 : Sseg(T,Gamma).dseg(T,g0,Gamma)+dseg(T,g0,Gamma)$
Sigma0 : matrix([sigma_z^2,0],[0,hbar^2/(4*sigma_z^2)])$
Sigmaf : S2.Sigma0.transpose(S2)$
overlap_exponent : transpose(delta).J.Sigmaf.transpose(J).delta/(2*hbar^2)$
contrast : exp(-overlap_exponent)$

print("Light-pulse arms: z in m and v in m/s")$
print("time       z_A          v_A          z_B          v_B")$
for row in [[0,xA0,xB0],[T,xAT,xBT],[2*T,xA2,xB2]] do
  printf(true,"~5,2f  ~12,6e ~12,6e ~12,6e ~12,6e~%",float(row[1]),
    subst(g0,grav,row[2][1,1]),subst(g0,grav,row[2][2,1]/m),
    subst(g0,grav,row[3][1,1]),subst(g0,grav,row[3][2,1]/m))$
print("Composed S(2T), d(2T), and propagated covariance Sigma_f")$
print(matrixmap('float,S2))$ print(matrixmap('float,d2))$
print(matrixmap('float,Sigmaf))$
printf(true,"phase = ~14,7e rad; gradient shift = ~12,5e rad~%",phase_value,gradient_shift)$
printf(true,"dPhi/dg = ~14,7e; closure = (~10,3e m, ~10,3e kg m/s)~%",
       scale_g,subst(g0,grav,delta[1,1]),subst(g0,grav,delta[2,1]))$
printf(true,"covariance-propagated contrast = ~12,9f~%",float(subst(g0,grav,contrast)))$
```

References

[1] Stephan Kleinert, Enno Kajari, Albert Roura, and Wolfgang P. Schleich. “Representation-free description of light-pulse atom interferometry including non-inertial effects”. In: *Physics Reports* 605

(2015), pp. 1–50. DOI: 10.1016/j.physrep.2015.09.004. URL: https://arxiv.org/abs/1512.00260.

[2] Philipp K. Schwartz and Domenico Giulini. "Post-Newtonian Hamiltonian description of an atom in a weak gravitational field". In: *Physical Review A* 100 (2019), p. 052116. DOI: 10.1103/PhysRevA.100.052116. URL: https://arxiv.org/abs/1908.06929.

[3] Albert Roura. "Gravitational redshift in quantum-clock interferometry". In: *Physical Review X* 10 (2020), p. 021014. DOI: 10.1103/PhysRevX.10.021014. URL: https://arxiv.org/abs/1810.06744.

Part IV

Asymptotic and spectral calculations

Chapter 20

Indexed WKB recurrences, roots, and quadrature

Abstract. An indexed Riccati recurrence illustrates generated symbolic objects, a logarithmic coordinate transformation, the symbolic-to-numerical handoff, bracketed turning points, and adaptive action integrals. The chapter distinguishes a formal asymptotic hierarchy from numerical convergence.

Keywords. Exact WKB, Riccati Recurrence, Turning Points, Root Finding, Adaptive Quadrature, Asymptotic Series

For exact WKB, resurgence, and radial extensions, see Refs. [1–3].

The radial quantity represented in the code

The screened potential is

$$V(r) = -\frac{ge^{-\alpha r}}{r} - iWe^{-\beta r}.$$

With the Langer replacement, the full radial curve used for turning points is

$$Q_{\text{full}}(r, E) = 2mr^2[V(r) - E] + \hbar^2(\ell + 1/2)^2.$$

The default $W = 0$ is a real reference problem. The loss term is present as an editable extension, but the turning-point routine still takes only the real part and is therefore not a complex-contour solver.

Why change to $x = \log(r/L)$

The assignment `r:L*exp(x)` inserts the logarithmic coordinate $r = Le^x$. In this coordinate the formal Riccati equation is

$$P^2 + \hbar P' = Q_0 + \hbar^2 Q_2,$$

where prime denotes d/dx,

$$Q_0 = 2mr^2[V(r) - E], \qquad Q_2 = (\ell + 1/2)^2,$$

and all other displayed Q_n vanish.

Expand

$$P = \sum_{n=0}^{N} \hbar^n S_n.$$

Matching powers gives

$$S_0 = \sqrt{Q_0}, \tag{20.1}$$

$$S_1 = -\frac{S_0'}{2S_0}, \tag{20.2}$$

$$S_n = \frac{Q_n - S_{n-1}' - sum_{j=1}^{n-1} S_j S_{n-j}}{2S_0}, \qquad n \geq 2. \tag{20.3}$$

This is exactly the loop in the driver. The arrays `Q[n]` and `S[n]` hold symbolic expressions, and `ratsimp` normalizes each recurrence result.

`r:L*exp(x)` assigns `r` globally. Later numerical functions use the dummy name `rr` to avoid colliding with that expression.

Turning points and the action

For a real reference energy, the two roots of $Q_{\text{full}} = 0$ delimit the classically allowed interval. The code contains two explicit brackets. It then evaluates

$$I(E) = \int_{r_-}^{r_+} \frac{\sqrt{-Q_{\text{full}}(r, E)}}{r}\, dr.$$

The factor (1/r) comes from (dx=dr/r); it is not an arbitrary normalization. The last table column is

$$\frac{I(E)}{\pi\hbar} - \frac{1}{2},$$

a semiclassical index diagnostic.

`quad_qags` returns the integral, estimated error, evaluation count, and status. The driver retains only `first(ans)` as the integral, so its printed table omits the other three return values.

Baseline output

At the probe radius $r = 4$, S_0 is imaginary. That is expected in the allowed region because $Q_0 < 0$ and the principal square root gives an imaginary momentum branch. The first four coefficients alternate between imaginary and real character in the baseline report.

As the scanned energy approaches zero from below, the outer turning point moves outward and the allowed action grows. Both trends are visible before any formal quantization condition is imposed.

Residuals of the generated hierarchy

The test does not compare long stored formulas. It substitutes each generated coefficient back into its defining residual:

$$R_n = 2S_0S_n + S'_{n-1} + sum_{j=1}^{n-1} S_j S_{n-j} - Q_n.$$

It evaluates R_n at three x-values and requires a small complex modulus. It also checks both root residuals for every energy and verifies positive actions. This residual-based approach continues to work if `Nwkb` changes.

Model scope. The series is formal and generally asymptotic; increasing `Nwkb` does not prove convergence. The fixed brackets assume exactly two real roots in stated intervals. For nonzero loss, complex turning points, Stokes continuation, contour deformation, and Borel summation require a dedicated continuation calculation.

Complete driver

Listing 20.1: Screened radial exact-WKB driver (`01_exact_wkb.mac`)

```
/* Screened radial exact-WKB workbench (Maxima 5.49). */
kill(all)$
display2d:false$ ratprint:false$ linel:200$

/* Editable physical parameters; W=0 gives the real reference problem. */
m:1$ hbar:7/20$ ell:1$ g:4$ alpha:1/4$
W:0$ beta:2/5$ L:1$ E:-1/5$ Nwkb:3$
langer:(ell+1/2)^2$
```

```
V(r):=-g*exp(-alpha*r)/r-%i*W*exp(-beta*r)$
Qfull(r,en):=2*m*r^2*(V(r)-en)+hbar^2*langer$

/* Langer-coordinate Riccati hierarchy: P^2+hbar P'=Q0+hbar^2 Q2. */
r:L*exp(x)$
Q[0]:2*m*r^2*(V(r)-E)$
for n:1 thru Nwkb do Q[n]:0$
Q[2]:langer$
S[0]:sqrt(Q[0])$
S[1]:-diff(S[0],x)/(2*S[0])$
for n:2 thru Nwkb do
  S[n]:ratsimp((Q[n]-diff(S[n-1],x)
        -sum(S[j]*S[n-j],j,1,n-1))/(2*S[0]))$
Pformal:sum(hbar^n*S[n],n,0,Nwkb)$

/* Real turning points and the allowed-region action for an energy en. */
turning_points(en):=[find_root(realpart(Qfull(rr,en)),rr,0.001,0.1),
                     find_root(realpart(Qfull(rr,en)),rr,0.1,30.0)]$
radial_action(en):=block([tp,ans],
  tp:turning_points(en),
  ans:quad_qags(sqrt(-realpart(Qfull(rr,en)))/rr,
                rr,tp[1],tp[2],epsrel=1.0e-8),
  [en,tp[1],tp[2],first(ans),bfloat(first(ans)/(%pi*hbar)-1/2)])$

energies:[-1/2,-7/20,-1/5,-1/10]$
action_table:map(radial_action,energies)$
rprobe:4.0$
coefficients_at_probe:makelist(
  [n,bfloat(ev(S[n],x=log(rprobe/L)))],n,0,Nwkb)$

print("SCREENED RADIAL EXACT-WKB MODEL")$
print("parameters [m,hbar,ell,g,alpha,W,beta] =",
      [m,hbar,ell,g,alpha,W,beta])$
print("columns [E,r_inner,r_outer,action,action/(pi*hbar)-1/2]")$
for row in action_table do print(row)$
print("Riccati coefficients [n,S_n] at r =",rprobe)$
for row in coefficients_at_probe do print(row)$
```

References

[1] Eric Delabaere and Frédéric Pham. “Resurgent methods in semi-classical asymptotics”. In: *Annales de l’Institut Henri Poincaré, Physique théorique* 71 (1999), pp. 1–94. URL: https://numdam.org/item/AIHPA_1999__71_1_1_0/.

[2] Katsushi Ito, Marcos Mariño, and Hongfei Shu. “TBA equations and resurgent quantum mechanics”. In: *Journal of High Energy Physics* 2019.1 (2019), p. 228. DOI: 10.1007/JHEP01(2019)228. URL: https://arxiv.org/abs/1811.04812.

[3] Okuto Morikawa and Shoya Ogawa. “Unified exact WKB framework for resonance—Zeldovich/complex-scaling regularization and rigged Hilbert space”. In: *Journal of High Energy Physics* 2025.10 (2025), p. 049. URL: https://arxiv.org/abs/2505.02301.

Chapter 21

Coupled threshold scattering with complex branches

Abstract. Exact Legendre integrals generate angular mixing before a two-channel threshold amplitude introduces nonanalytic logarithms, a closed channel, absorption, and branch-sensitive delays. Dimension tracking and sampled passivity checks keep the matrix construction interpretable.

Keywords. Threshold Scattering, Gaunt Coefficients, Effective Range, Closed Channel, Complex Branches, Time Delay

Threshold expansions and long-range scattering corrections are discussed in Refs. [1–3].

Exact angular mixing first

For (m=0) and $\ell \in \{0, 2, 4\}$, the code constructs

$$G_{\ell\ell'} = \frac{\sqrt{(2\ell+1)(2\ell'+1)}}{2} \int_{-1}^{1} P_\ell(z) P_2(z) P_{\ell'}(z)\, dz.$$

The variable is reset with `z:'z` so a previous assignment cannot enter the integral. `genmatrix` supplies the two angular indices, and `radcan` gives exact radicals. The output is

$$G = \begin{pmatrix} 0 & 1/\sqrt{5} & 0 \\ 1/\sqrt{5} & 2/7 & 6/(7\sqrt{5}) \\ 0 & 6/(7\sqrt{5}) & 20/77 \end{pmatrix}.$$

Selection rules are visible as exact zeros. The test checks three entries, including $G_{0,2} = 1/\sqrt{5}$.

Dimension map for the retained amplitude

The model keeps the first two partial waves. The dimensions are

Table 21.1: Dimensions and roles in the retained threshold amplitude

Object	Size	Role
`Acoef`, `Rcoef`, `Loss`	2×2	scattering-length, effective-range, and absorptive matrices
`Dtail`	2×2	leading block of the Gaunt matrix divided by 3
`Bclose`	2×1	coupling to one closed channel
`Pmat`, `Psqrt`	2×2 diagonal	threshold powers for $\ell = 0, 2$
`Finv`, `Famp`, `Smat`	2×2	inverse amplitude, amplitude, and scattering matrix

`submatrix(3,Gaunt,3)` deletes the third row and column, thereby retaining the leading 2×2 block.

The inverse amplitude

The programmed model is

$$f^{-1}(k) = -A^{-1} + \frac{k^2}{2}R + k^2\log(-ik)D - \frac{BB^T}{\gamma + \sqrt{\delta - k^2}} \tag{21.1}$$

$$- i[P(k) + L], \tag{21.2}$$

with $P = \mathrm{diag}(k, k^5)$. The terms represent the analytic effective-range part, a nonanalytic tail, a closed threshold, open-channel powers, and loss.

`cnum` applies `rectform`, extracts both parts, and converts each to a machine float. `nummat` lifts that conversion entry by entry. This boundary between exact construction and numerical inversion is intentional.

The scattering matrix is

$$S = I + 2iP^{1/2}fP^{1/2},$$

where $P^{1/2} = \mathrm{diag}(k^{1/2}, k^{5/2})$.

Decoding the mean-delay denominator

The code approximates a two-channel mean trace delay from $\log\det S$. A centered derivative in k supplies $2\,dk$; converting from k to $E = k^2$ supplies another $2k$; averaging over two retained channels supplies another factor 2. Thus

$$\bar{\tau}(k) \approx -i\frac{\log\det S(k + dk) - \log\det S(k - dk)}{8k\,dk}.$$

This compact formula is branch-sensitive. If $\arg\det S$ crosses the principal-log cut between samples, a $2\pi i$ jump contaminates the finite difference. Phase unwrapping replaces the principal arguments by a continuous sequence along the sampled path.

Computed observables and regression conditions

The real-momentum table prints (k), $|\det S|$, the real and imaginary mean delay, and $|\det f^{-1}|$. Loss makes the sampled determinant modulus less than one. The complex grid then prints a pole indicator: small $|\det f^{-1}|$ may guide a root search but is not itself a pole.

The test verifies exact Gaunt algebra, sampled $|\det S| \le 1$, numeric pole indicators, and consistent determinant helper functions. Determinant passivity is weaker than checking that every singular value of (S) is at most one.

Model scope. Principal `log` and `sqrt` do not perform scattering-sheet continuation. The complex scan is coarse, the delay phase is not unwrapped, and passivity is sampled only through a determinant proxy. Tail coefficients, coupled radial matching, and uncertainty propagation are inputs to future work.

Complete driver

Listing 21.1: Coupled threshold-amplitude driver (`03_threshold_universality.mac`)

```
/* Matrix threshold amplitude with dipolar Gaunt mixing and loss. */
kill(all)$
display2d:false$ ratprint:false$ linel:200$

/* Exact axial P2 coupling in the m=0 basis ell=0,2,4. */
ells:[0,2,4]$ z:'z$
Gaunt:genmatrix(lambda([i,j],radcan(
  sqrt((2*ells[i]+1)*(2*ells[j]+1))/2
  *integrate(legendre_p(ells[i],z)*legendre_p(2,z)
             *legendre_p(ells[j],z),z,-1,1))),3,3)$
```

```
11
12 /* Two retained waves: analytic ERE, k^2 log(-ik) tail, closed threshold. */
13 Acoef:matrix([3,2/5],[2/5,2])$
14 Rcoef:matrix([4/5,1/5],[1/5,3/5])$
15 Dtail:submatrix(3,Gaunt,3)/3$
16 Bclose:matrix([1/4],[1/7])$ delta:9/25$ gamma:1/2$
17 Loss:matrix([1/50,0],[0,1/100])$
18 Pmat(k):=matrix([k,0],[0,k^5])$
19 Psqrt(k):=matrix([sqrt(k),0],[0,k^(5/2)])$
20 Ltail(k):=k^2*log(-%i*k)*Dtail$
21 Lclosed(k):=-Bclose.transpose(Bclose)/(gamma+sqrt(delta-k^2))$
22 Finv(k):=-invert(Acoef)+k^2*Rcoef/2+Ltail(k)+Lclosed(k)
23           -%i*(Pmat(k)+Loss)$
24
25 cnum(q):=block([u:rectform(q)],
26   float(realpart(u))+%i*float(imagpart(u)))$
27 nummat(M):=genmatrix(lambda([i,j],cnum(M[i,j])),length(M),length(M[1]))$
28 Famp(k):=nummat(invert(Finv(k)))$
29 Smat(k):=nummat(ident(2)+2*%i*Psqrt(k).Famp(k).Psqrt(k))$
30 detS(k):=cnum(determinant(Smat(k)))$
31 dk:1.0e-4$
32 mean_delay(k):=cnum(-%i*(log(detS(k+dk))-log(detS(k-dk)))/(8*k*dk))$
33 observable_row(k):=block([S:Smat(k),tau:mean_delay(k)],
34   [k,cabs(determinant(S)),realpart(tau),imagpart(tau),
35    cabs(cnum(determinant(Finv(k))))])$
36
37 k_scan:[0.04,0.07,0.10,0.14,0.20]$
38 observables:map(observable_row,k_scan)$
39 pole_grid:[0.04-%i*0.03,0.08-%i*0.03,0.12-%i*0.03,
40            0.04-%i*0.07,0.08-%i*0.07,0.12-%i*0.07]$
41 pole_scan:makelist([kp,cabs(cnum(determinant(Finv(kp))))],kp,pole_grid)$
42 print("MATRIX THRESHOLD MODEL; exact Gaunt block =",Gaunt)$
43 print("columns [k,|det S|,Re tau,Im tau,|det f^{-1}|]")$
44 for row in observables do print(row)$
45 print("representative S(k=0.10) =",Smat(0.10))$
46 print("complex-k pole indicator [k,|det f^{-1}|]")$
47 for row in pole_scan do print(row)$
```

References

[1] T. F. O'Malley, L. Spruch, and L. Rosenberg. "Modification of effective-range theory in the presence of a long-range inverse-fourth-power potential". In: *Journal of Mathematical Physics* 2 (1961), pp. 491–498. DOI: 10.1063/1.1703735.

[2] H.-W. Hammer and Dean Lee. "Causality and the effective range expansion". In: *Annals of Physics* 325 (2010), pp. 2212–2233. DOI: 10.1016/j.aop.2010.06.006. URL: https://arxiv.org/abs/1002.4603.

[3] Zbigniew Idziaszek and Paul S. Julienne. "Universal rate constants for reactive collisions of ultracold molecules". In: *Physical Review Letters* 104 (2010), p. 113202. DOI: 10.1103/PhysRevLett.104.113202. URL: https://arxiv.org/abs/0912.0370.

Chapter 22

Floquet–Sambe transfer matrices

Abstract. Floquet sidebands expand a two-component transfer problem into dynamically sized Sambe blocks. The chapter develops Toeplitz harmonics, contact jumps, removable propagation limits, characteristic roots, Bloch multipliers, and truncation diagnostics.

Keywords. Floquet Theory, Sambe Space, Transfer Matrix, Toeplitz Matrix, Bloch Multiplier, Truncation

General Floquet and non-Hermitian Floquet background can be found in Refs. [1–3].

From one channel to Sambe space

A harmonic cutoff Q retains sidebands $q = -Q, \ldots, Q$, so $N_s = 2Q + 1$. At $Q = 0$ the transfer matrix is the familiar 2×2 position-derivative map. At the default $Q = 1$, $N_s = 3$ and the full matrix is 6×6.

`setup_sambe(qcut)` assigns five related globals: `Q`, `qs`, `Ns`, `Iq`, and `Zq`. The paired test calls it again with $Q = 2$ and verifies a 10×10 transfer matrix.

Harmonics become a Toeplitz block

Each contact stores zero, $+1$, and -1 Fourier coefficients in a list. For sideband indices q_i, q_j, the matrix entry depends only on the difference $d = q_i - q_j$:

$$G_{ij} = g_0\delta_{d0} + g_{+1}\delta_{d1} + g_{-1}\delta_{d,-1}.$$

Unequal g_{+1} and the conjugate of g_{-1} allow a non-Hermitian contact. The delta jump is the block matrix

$$J(G) = \begin{pmatrix} I & 0 \\ 2mG/\hbar^2 & I \end{pmatrix}.$$

Free propagation and the removable limit

For quasienergy ϵ, sideband momenta are

$$k_q = \sqrt{2m(\epsilon + q\omega)}/\hbar.$$

The diagonal free blocks contain $\cos(k_q s)$, $\sin(k_q s)/k_q$, and $-k_q \sin(k_q s)$. The helper `sine_kernel` evaluates the middle expression by a limit so it returns s at $k_q = 0$.

The assembled free transfer is

$$F(s) = \begin{pmatrix} C & U \\ V & C \end{pmatrix}.$$

Reading matrix products from right to left, the cell transfer

$$M = F(b)J_B F(a)J_A$$

means jump at A, propagate distance a, jump at B, then propagate distance b.

Bloch multipliers

The code constructs $p(\lambda) = \det(\lambda I - M)$ with `charpoly`, expands it, and applies `allroots`. Since roots are returned as equations, `map(rhs,...)` extracts their right sides. The report sorts multiplier moduli and prints the full complex set at one quasienergy.

Every baseline cell determinant is one to about 10^{-16}, and the product of all characteristic roots agrees with the determinant. These are structural checks on the transfer construction. Individual moduli need not be one in the non-Hermitian model.

Truncation and conditioning

Increasing Q changes every matrix dimension. A trustworthy result needs a cutoff sequence for observables, not only a successful $Q = 2$ construction. Moreover, forming a characteristic polynomial and finding all of its roots can become ill-conditioned as dimension grows. The compact method is appropriate to this small demonstration, not automatically to a production Sambe solve.

The cutoff is changed through `setup_sambe`, which updates every dimension-dependent object; changing `Q` alone leaves an inconsistent state.

Model scope. The driver retains only three contact harmonics and a small finite sideband space. Principal square roots fix channel-momentum branches. There is no large-(Q) stabilization, generalized-Brillouin-zone continuation, Evans function, or cutoff-convergence certificate.

Complete driver

Listing 22.1: Finite-Sambe transfer driver (`04_floquet_dirac_comb.mac`)

```
/* Finite-Sambe transfer spectrum for a driven non-Hermitian delta comb. */
kill(all)$
display2d:false$ ratprint:false$ linel:200$
m:1$ hbar:1$ omega:1$ a:9/20$ b:11/20$

/* Harmonic contact data; unequal +/- coefficients allow non-Hermiticity. */
gA:[1/3,1/5+%i/9,1/7-%i/11]$
gB:[-1/4,-2/9+%i/13,-1/6-%i/10]$
setup_sambe(qcut):=(Q:qcut,qs:makelist(q,q,-Q,Q),
  Ns:2*Q+1,Iq:ident(Ns),Zq:zeromatrix(Ns,Ns))$
setup_sambe(1)$

join4(A,B,C,D):=addrow(addcol(A,B),addcol(C,D))$
toeplitz_harmonic(g):=genmatrix(lambda([i,j],block([d:qs[i]-qs[j]],
  g[1]*kron_delta(d,0)+g[2]*kron_delta(d,1)
  +g[3]*kron_delta(d,-1))),Ns,Ns)$
jump(g):=join4(Iq,Zq,2*m*toeplitz_harmonic(g)/hbar^2,Iq)$
kq(en,q):=sqrt(2*m*(en+q*omega))/hbar$
sine_kernel(k,s):=block([x],limit(sin(x*s)/x,x,k))$

free_transfer(en,s):=block([C,U,V],
  C:genmatrix(lambda([i,j],kron_delta(i,j)*cos(kq(en,qs[i])*s)),Ns,Ns),
  U:genmatrix(lambda([i,j],kron_delta(i,j)
      *sine_kernel(kq(en,qs[i]),s)),Ns,Ns),
  V:genmatrix(lambda([i,j],-kron_delta(i,j)*kq(en,qs[i])
      *sin(kq(en,qs[i])*s)),Ns,Ns), join4(C,U,V,C))$
cell_transfer(en):=bfloat(free_transfer(en,b).jump(gB)
                          .free_transfer(en,a).jump(gA))$

floquet_multipliers(en):=block([lam:'lam,p,roots],
  p:expand(charpoly(cell_transfer(en),lam)),
  roots:map(rhs,allroots(p)),roots)$
multiplier_row(en):=block([M:cell_transfer(en),r],
```

```
  r:floquet_multipliers(en),
  [en,cabs(rectform(determinant(M)-1)),sort(map(cabs,r))])$

eps_scan:[0.35,0.55,0.75]$
spectrum_table:map(multiplier_row,eps_scan)$
roots_mid:floquet_multipliers(0.55)$
print("FLOQUET-SAMBE DELTA-COMB MODEL: Q =",Q,"sidebands =",qs)$
print("contact Toeplitz block A =",toeplitz_harmonic(gA))$
print("columns [quasienergy, |det(M)-1|, sorted |Bloch multipliers|]")$
for row in spectrum_table do print(row)$
print("multipliers [beta,|beta|] at quasienergy 0.55")$
for root in roots_mid do print([root,cabs(root)])$
```

References

[1] Marin Bukov, Luca D'Alessio, and Anatoli Polkovnikov. "Universal high-frequency behavior of periodically driven systems: from dynamical stabilization to Floquet engineering". In: *Advances in Physics* 64 (2015), pp. 139–226. DOI: 10.1080/00018732.2015.1055918. URL: https://arxiv.org/abs/1407.4803.

[2] Mark S. Rudner and Netanel H. Lindner. "Band structure engineering and non-equilibrium dynamics in Floquet topological insulators". In: *Nature Reviews Physics* 2 (2020), pp. 229–244. DOI: 10.1038/s42254-020-0170-z. URL: https://arxiv.org/abs/1909.02008.

[3] Longwen Zhou and Da-Jian Zhang. *Non-Hermitian Floquet topological matter: a review*. 2023. URL: https://arxiv.org/abs/2305.16153 (visited on 07/16/2026).

Part V

Capstone: a discretized inverse problem

Chapter 23

Matrix Marchenko reconstruction and a custom complex solver

Abstract. The capstone discretizes a coupled Marchenko integral equation, flattens channel and grid indices, solves several complex right-hand sides with an explicit Gaussian-elimination helper, and reconstructs a potential by a finite difference. Its limitations make pivoting, conditioning, cutoff, and grid studies unavoidable.

Keywords. Inverse Scattering, Marchenko Equation, Nystrom Method, Complex Linear System, Gaussian Elimination, Potential Reconstruction

For matrix inverse scattering, non-Hermitian spectral structure, and reconstruction theory, see Refs. [1–4].

The inverse equation

The two-channel Marchenko equation is represented as

$$K(x,y) + F(x+y) + \int_x^\infty K(x,s)F(s+y)\,ds = 0.$$

The kernel

$$F(s) = (C_0 + sC_1)e^{-\kappa s}$$

contains an $se^{-\kappa s}$ term, the signature used here for a double Jost pole. The residue matrices are complex and need not commute or be Hermitian.

The integral is truncated at `cutoff` and sampled on `ngrid` points from x_0 to that cutoff. Trapezoidal endpoint weights are constructed with Kronecker deltas.

Flattening two indices

Each unknown has a grid index $j = 1, \ldots, N$ and channel index $b = 1, 2$. The code flattens them as

$$(j,b) \longmapsto 2(j-1) + b.$$

For $N = 3$, the complete mapping is shown in Figure 23.1. Consecutive channel entries remain adjacent while advancing the grid index moves to the next pair.

Given a flat row, it recovers

```
j : floor((row-1)/2)+1$
b : mod(row-1,2)+1$
```

and uses the same mapping for a column pair (l, c). The discretized matrix is

$$A_{(j,b),(l,c)} = \delta_{jl}\delta_{bc} + w_l F_{cb}(s_l + y_j).$$

The order F_{cb}, visible as `[c,b]`, follows the matrix product in the integral equation; reversing it changes the model.

For each output channel a, the right-hand side is $-F_{ab}(x_0 + y_j)$. Thus the same $2N \times 2N$ matrix is solved against two right-hand sides.

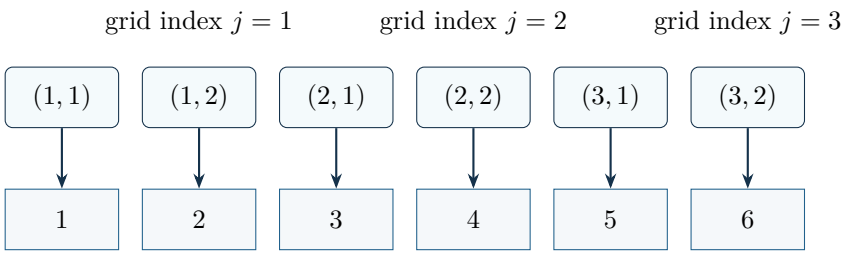

Figure 23.1: Flattening a grid index and a two-valued channel index into one linear-system index.

Why the helper stores real–imaginary pairs

The library converts each complex float z to the pair $[\operatorname{Re} z, \operatorname{Im} z]$. It then defines

$$(a,b) + (c,d) = (a+c, b+d), \tag{23.1}$$

$$(a,b)(c,d) = (ac - bd, ad + bc), \tag{23.2}$$

$$\frac{(a,b)}{(c,d)} = \left(\frac{ac+bd}{c^2+d^2}, \frac{bc-ad}{c^2+d^2}\right). \tag{23.3}$$

This makes every floating operation explicit and avoids depending on an external linear-algebra package. `AA` and `bb` are converted copies, so elimination does not mutate the caller's symbolic matrices.

Forward elimination and back substitution cost $O(n^3)$. There is no row pivoting, zero-pivot check, or condition estimate. That omission is the central numerical limitation of the capstone, not a minor implementation detail.

Recovering the diagonal kernel

The first grid point is $y_1 = x_0$. After solving both right-hand sides, `u[a][b,1]` selects the element corresponding to $K_{ab}(x_0, x_0)$. The potential is related to the diagonal kernel by

$$V(x) = -2\frac{d}{dx}K(x,x).$$

A centered derivative gives

$$-2\frac{K(x+dx, x+dx) - K(x-dx, x-dx)}{2dx} = -\frac{K_+ - K_-}{dx},$$

which explains the apparent absence of a factor two in the code.

The baseline regression

For $x = 1$, the test solves each right-hand side and requires the matrix residual $Au - b$ to be below 10^{-8} entry by entry. It also ensures that the reconstructed potential has not collapsed to zero. These are necessary implementation checks, but a small residual can coexist with a large solution error when A is ill-conditioned.

Model scope. The $N = 6$ calculation is illustrative. It has no pivoting, conditioning estimate, continuum-data term, grid/cutoff/step convergence, inverse-data regularization, or uniqueness analysis. A small algebraic residual does not certify a stable inverse reconstruction. Its grid definition requires $x_0 <$`cutoff`.

Complete numerical helper

Listing 23.1: Dependency-free complex Gaussian solver (`lib/02_marchenko_numeric.mac`)

```
/* Small complex Gaussian solver using [real,imaginary] float pairs. */
cpair(z):=block([q:rectform(z)],
  [float(realpart(q)),float(imagpart(q))])$
cfrom(p):=p[1]+%i*p[2]$
cadd(p,q):=[p[1]+q[1],p[2]+q[2]]$
csub(p,q):=[p[1]-q[1],p[2]-q[2]]$
cmul(p,q):=[p[1]*q[1]-p[2]*q[2],p[1]*q[2]+p[2]*q[1]]$
cdiv(p,q):=block([d:q[1]^2+q[2]^2],
  [(p[1]*q[1]+p[2]*q[2])/d,(p[2]*q[1]-p[1]*q[2])/d])$

complex_gauss(A,b):=block([n,AA,bb,f,u,acc],
  n:length(A),
  AA:genmatrix(lambda([i,j],cpair(A[i,j])),n,n),
  bb:makelist(cpair(b[i,1]),i,1,n),
  for k:1 thru n-1 do for i:k+1 thru n do (
    f:cdiv(AA[i,k],AA[k,k]),
    for j:k thru n do AA[i,j]:csub(AA[i,j],cmul(f,AA[k,j])),
    bb[i]:csub(bb[i],cmul(f,bb[k]))),
  u:makelist([0.0,0.0],i,1,n), u[n]:cdiv(bb[n],AA[n,n]),
  for i:n-1 step -1 thru 1 do (
    acc:[0.0,0.0],
    for j:i+1 thru n do acc:cadd(acc,cmul(AA[i,j],u[j])),
    u[i]:cdiv(csub(bb[i],acc),AA[i,i])),
  genmatrix(lambda([i,j],cfrom(u[i])),n,1))$
```

Complete driver

Listing 23.2: Two-channel Marchenko driver (`02_nonhermitian_inverse_scattering.mac`)

```
/* Two-channel higher-order-pole Marchenko reconstruction. */
kill(all)$
display2d:false$ linel:200$
ratprint:false$
batchload("maxima/lib/02_marchenko_numeric.mac")$
numeric_matrix(M):=genmatrix(lambda([i,j],cfrom(cpair(M[i,j]))),
  length(M),length(M[1]))$

/* C1 multiplies s exp(-kappa s), the signature of a double Jost pole. */
kappa:6/5$
C0:matrix([3/5,1/5+%i/10],[-1/8+%i/12,2/5])$
C1:matrix([1/6,-1/9+%i/14],[1/10+%i/15,-1/7])$
F(s):=(C0+s*C1)*exp(-kappa*s)$

/* Nystr\"om grid for K(x,y)+F(x+y)+int K(x,s)F(s+y) ds=0. */
ngrid:6$ cutoff:7$ dx:0.02$
marchenko_system(x0):=block([h,y,s,w,A],
  h:(cutoff-x0)/(ngrid-1),
  y:makelist(x0+(j-1)*h,j,1,ngrid), s:y,
  w:makelist(h*(1-(kron_delta(j,1)+kron_delta(j,ngrid))/2),
             j,1,ngrid),
  A:genmatrix(lambda([row,col],block([j,b,l,c],
    j:floor((row-1)/2)+1, b:mod(row-1,2)+1,
    l:floor((col-1)/2)+1, c:mod(col-1,2)+1,
    kron_delta(j,l)*kron_delta(b,c)+w[l]*F(s[l]+y[j])[c,b])),
    2*ngrid,2*ngrid),
  [y,s,w,A])$

rhs_vector(x0,a):=genmatrix(lambda([row,col],block([j,b],
  j:floor((row-1)/2)+1, b:mod(row-1,2)+1,
```

```
  -F(x0+(x0+(j-1)*(cutoff-x0)/(ngrid-1)))[a,b])),
  2*ngrid,1)$

Kdiagonal(x0):=block([sys,A,u],
  sys:marchenko_system(x0), A:sys[4],
  u:makelist(complex_gauss(A,rhs_vector(x0,a)),a,1,2),
  genmatrix(lambda([a,b],u[a][b,1]),2,2))$
Vreconstructed(x0):=numeric_matrix(
  -(Kdiagonal(x0+dx)-Kdiagonal(x0-dx))/dx)$

x_samples:[2/5,1,8/5]$
profile:makelist([xx,Kdiagonal(xx),Vreconstructed(xx)],xx,x_samples)$
print("TWO-CHANNEL MARCHENKO MODEL")$
print("higher-order kernel F(1) =",numeric_matrix(F(1)))$
print("columns: x, K(x,x), reconstructed V(x)")$
for row in profile do (
  print("x =",row[1]), print("K =",row[2]), print("V =",row[3]))$
```

References

[1] G. Freiling and V. Yurko. “An inverse problem for the non-selfadjoint matrix Sturm–Liouville equation on the half-line”. In: *Journal of Inverse and Ill-Posed Problems* 15 (2007), pp. 785–798. DOI: 10.1515/jiip.2007.042.

[2] Natalia Bondarenko. *An inverse spectral problem for the matrix Sturm–Liouville operator on the half-line*. 2015. URL: https://arxiv.org/abs/1412.5962 (visited on 07/16/2026).

[3] Natalia Bondarenko. “Inverse scattering on the line for the matrix Sturm–Liouville equation”. In: *Journal of Differential Equations* 262 (2017), pp. 2073–2105. DOI: 10.1016/j.jde.2016.10.040. URL: https://arxiv.org/abs/1512.06210.

[4] Ali Mostafazadeh. “Physics of spectral singularities”. In: *Geometric Methods in Physics* (2015), pp. 145–165. URL: https://arxiv.org/abs/1412.0454.

Chapter A

Maxima quick reference for these programs

Evaluation and data

Table A.1: Evaluation, control-flow, and data constructs

Construct	Meaning in this project
`x:value`	immediate assignment
`f(x):=body`	delayed function definition
`left=right`	equation or replacement rule, not assignment
`expr;` / `expr$`	evaluate with / without automatic display
`'name`	suppress evaluation of a name for this step
`values`, `functions`	inspect names and function definitions stored in the session
`kill(name)`, `kill(all)`	remove one definition or reset the working session
`describe("topic")`	open Maxima's installed help for a command or concept
`[a,b,c]`	one-based list
`M[i,j]`	one-based matrix entry
`Q[n]`	indexed symbolic variable unless `Q` is a list
`block([a,b],...)`	local variables; final expression is the result
`map(f,L)`	apply an existing function to every list element
`makelist(expr,i,a,b)`	construct a list from an indexed expression
`lambda([i,j],expr)`	anonymous function, often used by `genmatrix`
`if c then a else b`	conditional expression; `c` must be true or false
`for i:a thru b do ...`	counted loop
`for i:a next e while c do ...`	loop with an explicit next value and stopping condition
`for x in L do ...`	loop over list elements
`true`, `false`, `and`, `or`, `not`	Boolean values and operators
`length`, `first`, `rest`, `sort`	common list queries and transformations
`floor(x)`, `mod(a,b)`	integer quotient and remainder helpers
`kron_delta(i,j)`	Kronecker delta used in generated matrices

Algebra and calculus

Table A.2: Algebra and symbolic-calculus constructs

Construct	Primary use
`expand(expr)`	expose polynomial terms
`factor(expr)`	expose polynomial factors
`coeff(expr,x,n)`	coefficient of x^n
`ratsimp(expr)`	rational simplification
`radcan(expr)`	radicals and rational powers
`trigsimp(expr)`	trigonometric identities
`diff(expr,x,n)`	derivative of order n
`integrate(expr,x,a,b)`	exact definite integral when available
`limit(expr,x,value)`	symbolic limit, including `inf`
`laplace(expr,t,z)`	Laplace transform
`ode2(ode,y,x)`	solve a supported first- or second-order ordinary differential equation
`ic1(sol,x=x0,y=y0)`	impose one initial condition on a first-order ODE solution
`taylor(expr,x,x0,n)`	local Taylor series through order n
`sum(expr,i,a,b)`, `product(...)`	symbolic finite or formal constructions
`solve(eq,x)`, `linsolve(eqs,vars)`	algebraic and linear equation solving
`assume`, `facts`, `forget`	manage global symbolic assumptions
`subst([x=a,y=b],expr)`	simultaneous-style list of replacements
`subst(a,x,expr)`	replace symbol x by value a
`ev(expr,x=a)`	evaluate under temporary assignments

Special functions

Table A.3: Special functions used in physics

Construct	Physical use
`gamma(x)`, `beta(a,b)`	factorial continuation and beta integrals
`erf(x)`, `erfc(x)`	integrated Gaussian weight and tails
`bessel_j`, `bessel_y`	cylindrical and radial waves
`bessel_i`, `bessel_k`	modified radial equations
`hankel_1`, `hankel_2`	outgoing and incoming cylindrical-wave combinations
`spherical_hankel1`, `spherical_hankel2`	spherical radiation-wave combinations
`legendre_p`, `assoc_legendre_p`	angular separation and multipoles
`hermite`, `gen_laguerre`	oscillator and Coulomb polynomial bases
`airy_ai`, `airy_bi`	simple turning-point solutions
`hypergeometric(A,B,z)`	generalized hypergeometric representation
`elliptic_kc(m)`, `elliptic_ec(m)`	complete elliptic integrals and nonlinear periods

Complex and numerical work

Table A.4: Complex and numerical constructs

Construct	Primary use

`%i`, `%pi`, `inf`	built-in imaginary unit, pi, infinity
`rectform(z)`	rewrite toward real plus imaginary form
`realpart(z)`, `imagpart(z)`	component extraction
`cabs(z)`	complex modulus
`conjugate(z)`	complex conjugation
`float(expr)`	machine floating evaluation
`bfloat(expr)`	arbitrary-precision floating evaluation
`fpprec:n`	set bigfloat precision to n decimal digits
`find_root(f,x,a,b)`	bracketed numerical root
`quad_qags(...)`	adaptive quadrature; returns a result list
`allroots(p)`	numerical polynomial roots as equations
`rhs(eq)`	right side of an equation returned by a solver

Scientific plotting

Table A.5: Plotting and data-export constructs

Construct	Primary use
`plot2d(expr,[x,a,b])`	explicit two-dimensional function plot
`[discrete,points]`	discrete coordinate pairs
`[style,points,lines]`	per-series point and line encodings
`[xlabel,"..."]`, `[ylabel,"..."]`	axis quantities and units
`gnuplot_term`	vector PDF rendering through gnuplot
`gnuplot_out_file`	stable output filename
`with_stdout`	explicit numerical data export

Matrix work

Table A.6: Matrix constructs

Construct	Primary use
`matrix([..],[..])`	explicit matrix
`genmatrix(f,m,n)`	generate entries by row and column
`ident(n)`, `zeromatrix(m,n)`	identity and zero matrices
`A.B`	matrix multiplication
`s*A`	scalar multiplication
`transpose(A)`	transpose without conjugation
`conjugate(transpose(A))`	Hermitian adjoint of a complex matrix
`determinant(A)`, `invert(A)`	small symbolic determinant or inverse
`submatrix(i,A,j)`	delete row i and column j
`addcol`, `addrow`	horizontal and vertical block assembly
`charpoly(A,x)`	characteristic polynomial
`matrixmap('float,A)`	apply a function to each matrix entry

Tests and output

Table A.7: Testing and output constructs

Construct	Primary use
`is(statement)`	ask whether a predicate is true
`equal(a,b)`	symbolic mathematical equality query
`numberp(expr)`	true when an expression is explicitly numeric
`error("message",data)`	abort a batch run with context
`print(...)`	human-readable Maxima values
`printf(...)`	controlled fixed/scientific fields
`batchload("path")`	execute a dependency in the current session

Common symptoms and causes

Table A.8: Common Maxima symptoms and their likely causes

Symptom	Likely cause or Maxima behavior
Expression stays unevaluated	misspelled function, missing assumptions, or a symbol that has no numeric value
Unexpected old value appears	a global immediate assignment remains active; `values`, `functions`, and `kill(all)` expose or clear that state
Matrix product is wrong	`*` was used instead of `.`, or the inner dimensions and index order are incompatible
Real/imaginary part stays complicated	the expression has not been converted to rectangular form with `rectform`
Equality test is unknown	the residual is not in a canonical form, or the inputs are approximate rather than exact
Root finder fails	the bracket has no sign change or the function is not numeric throughout the interval
Sudden jump in a complex scan	a principal branch cut lies between samples
High-precision result does not improve	machine decimals entered before the bigfloat conversion
Test passes but result changes with grid	the regression identity holds while the discretized observable remains resolution-dependent

Chapter B

The complete regression suite

The following listings are the executable tests paired with the twelve drivers. They batch-load the corresponding source and evaluate exact identities, limiting cases, numerical residuals, and structural invariants.

WKB hierarchy, roots, and actions

Listing B.1: Regression test for Model 01

```
/* Regression tests for 01_exact_wkb.mac. */
batchload("maxima/01_exact_wkb.mac")$
zero_p(z):=is(cabs(rectform(bfloat(z)))<1.0e-7)$

xchecks:[-1,0,1]$
r1:2*S[0]*S[1]+diff(S[0],x)-Q[1]$
for xv in xchecks do
  if not zero_p(ev(r1,x=xv)) then error("first Riccati coefficient")$
for n:2 thru Nwkb do block([rn],
  rn:2*S[0]*S[n]+diff(S[n-1],x)
     +sum(S[j]*S[n-j],j,1,n-1)-Q[n],
  for xv in xchecks do
    if not zero_p(ev(rn,x=xv)) then error("Riccati recurrence",n,xv))$
for row in action_table do (
  if not zero_p(Qfull(row[2],row[1])) then error("inner root regression"),
  if not zero_p(Qfull(row[3],row[1])) then error("outer root regression"),
  if not is(row[4]>0) then error("action sign regression"))$
print("TEST 01 OK: hierarchy, turning points, and actions")$
```

Marchenko linear solve and reconstruction

Listing B.2: Regression test for Model 02

```
/* Regression tests for 02_nonhermitian_inverse_scattering.mac. */
batchload("maxima/02_nonhermitian_inverse_scattering.mac")$
matrix_small_p(M,tol):=block([ok:true,p],
  for i:1 thru length(M) do
    for j:1 thru length(M[i]) do (
      p:cpair(M[i,j]),
      if not is(sqrt(p[1]^2+p[2]^2)<tol) then ok:false), ok)$

xcheck:1$ sys:marchenko_system(xcheck)$ A:sys[4]$
for a:1 thru 2 do block([rhs,u],
  rhs:rhs_vector(xcheck,a), u:complex_gauss(A,rhs),
  if not matrix_small_p(A.u-rhs,1.0e-8) then
    error("Marchenko linear-system regression"))$
if matrix_small_p(Vreconstructed(1),1.0e-6) then
  error("reconstructed potential collapsed to zero")$
print("TEST 02 OK: block solve and reconstructed potential")$
```

Threshold angular algebra and passivity proxy

Listing B.3: Regression test for Model 03

```
/* Regression tests for 03_threshold_universality.mac. */
batchload("maxima/03_threshold_universality.mac")$
zero_exact_p(q):=is(radcan(q)=0)$
if not zero_exact_p(Gaunt[1,3]) then error("Gaunt selection rule")$
if not zero_exact_p(Gaunt[1,2]-1/sqrt(5)) then error("Gaunt normalization")$
if not zero_exact_p(Gaunt[2,3]-6*sqrt(5)/35) then
  error("Gaunt ell=2 to ell=4 coupling")$
for row in observables do (
  if not is(row[2]<=1.0+1.0e-8) then error("passivity regression"),
  if not numberp(row[5]) then error("pole indicator is not numeric"))$
if not is(cabs(cnum(determinant(Smat(0.1))-detS(0.1)))<1.0e-10) then
  error("S determinant regression")$
print("TEST 03 OK: Gaunt algebra, passive S matrix, and pole scan")$
```

Floquet determinant, roots, and cutoff dimension

Listing B.4: Regression test for Model 04

```
/* Regression tests for 04_floquet_dirac_comb.mac. */
batchload("maxima/04_floquet_dirac_comb.mac")$
for row in spectrum_table do
  if not is(row[2]<1.0e-8) then error("cell determinant regression")$
if not is(length(roots_mid)=2*Ns) then error("multiplier count regression")$
root_product:1$
for r in roots_mid do root_product:root_product*r$
if not is(cabs(rectform(root_product-determinant(cell_transfer(0.55))))
          <1.0e-6) then error("multiplier product regression")$
setup_sambe(2)$
M_Q2:cell_transfer(0.55)$
if not is(length(M_Q2)=10) then error("arbitrary-Q dimension regression")$
if not is(cabs(rectform(determinant(M_Q2)-1))<1.0e-8) then
  error("Q=2 determinant regression")$
print("TEST 04 OK: spectrum, multiplier product, and Q=2 construction")$
```

Dephasing derivatives, limit, and Laplace transform

Listing B.5: Regression test for Model 05

```
/* Regression tests for the finite-coupling dephasing model. */
batchload("maxima/05_nonmarkovian_zeno.mac")$

r1:ratsimp(diff(Goh(t),t)-eta*wc^2*t/(1+wc^2*t^2))$
r2:ratsimp(diff(Gstr(t),t)-sigma2*tc*(1-exp(-t/tc)))$
inner_limit:limit(lam^2*Gtot(tau/lam),lam,inf)$
inner_target:(eta*wc^2+sigma2)*tau^2/2$
r3:radcan(inner_limit-inner_target)$
r4:radcan(Cstr_tilde(z)-laplace(sigma2*exp(-t/tc),t,z))$

if not is(equal(r1,0)) then error("Ohmic rate regression",r1)$
if not is(equal(r2,0)) then error("structured rate regression",r2)$
if not is(equal(r3,0)) then error("inner expansion regression",r3)$
if not is(equal(r4,0)) then error("Laplace kernel regression",r4)$
if not is(float(coherence(4,1/2)) < float(coherence(2,1/2))) then
  error("finite-coupling ordering regression")$
print("TEST 05 OK")$
```

Efimov cycle and ladder

Listing B.6: Regression test for Model 06

```
/* Regression tests for the complex Efimov model. */
batchload("maxima/06_efimov_corrections.mac")$

Hs(x,s):=cos(x+atan(s))/cos(x-atan(s))$
r1:trigsimp(Hs(x+%pi,s)-Hs(x,s))$
r2:radcan(kappaLO(n+1)/kappaLO(n)-exp(-%pi/s0))$
r3:radcan(ev(kappaNLO(n)-kappaLO(n),re=0,kso=0))$
cycle_error:abs(float(rectform(Hrun(3*lambda0)-Hrun(3))))$

if not is(equal(r1,0)) then error("limit-cycle identity regression",r1)$
if not is(equal(r2,0)) then error("LO ladder regression",r2)$
if not is(equal(r3,0)) then error("NLO switch-off regression",r3)$
if not is(cycle_error < 1.0e-12) then error("complex cycle regression",cycle_error)$
for n:0 thru 3 do
  if not is(float(width(n)) > 0) then error("width-sign regression",n)$
print("TEST 06 OK")$
```

DFT normalization, potential, scaling, and response

Listing B.7: Regression test for Model 07

```
/* Regression tests for the orbital-free Gaussian model. */
batchload("maxima/07_orbital_free_dft.mac")$

norm:integrate(4*%pi*r^2*nrad(r),r,0,inf)$
r1:radcan(norm-Ne)$
nr:nrad(r)$
vW_direct:ratsimp((diff(nr,r)^2/nr^2
                  -2*(diff(nr,r,2)+2*diff(nr,r)/r)/nr)/8)$
r2:ratsimp(vW_direct-vW(r))$
r3:ratsimp(Ts(gam^2*a)-gam^2*Ts(a))$
r4:ratsimp(Kmodel(q)-KLindhard2(q))$

if not is(equal(r1,0)) then error("normalization regression",r1)$
if not is(equal(r2,0)) then error("Euler potential regression",r2)$
if not is(equal(r3,0)) then error("coordinate scaling regression",r3)$
if not is(equal(r4,0)) then error("linear response regression",r4)$
print("TEST 07 OK")$
```

Dirac dispersion, matching roots, and degeneracy

Listing B.8: Regression test for Model 08

```
/* Regression tests for the deformed supercritical Dirac ladder. */
batchload("maxima/08_supercritical_dirac.mac")$

r1:radcan(kbase(n+1)/kbase(n)-exp(-%pi/beta))$
r2:radcan(Elevel(2,1)^2-Delta^2-(hbarv*kcorr(2,1))^2)$
r3:radcan(Dmatch(Elevel(1,1),1))$
r4:radcan(Dmatch(Elevel(1,-1),1))$
degenerate:float(rectform(ev(Elevel(1,1)-Elevel(1,-1),
                             cIntervalley=0)))$

if not is(equal(r1,0)) then error("ideal ladder regression",r1)$
if not is(equal(r2,0)) then error("Dirac dispersion regression",r2)$
if not is(equal(r3,0) and equal(r4,0)) then
  error("matching determinant regression",r3,r4)$
if not is(abs(degenerate) < 1.0e-12) then
```

```
  error("valley-degeneracy regression",degenerate)$
for n:0 thru 3 do
  if not is(float(width(n,1)) > 0 and float(width(n,-1)) > 0) then
    error("width-sign regression",n)$
print("TEST 08 OK")$
```

Coevolution and covariance

Listing B.9: Regression test for Model 09

```
batchload("maxima/09_tensor_force_eft.mac")$
tol : 1e-10$
for sv in [0,0.25,0.5,1,2,4] do
  if abs(float(expect(psi(sv),Jflow(sv))-expect(psi0,J0))) > tol
  then error("09 coevolution regression failed at s =",sv)$
if abs(float(H(4)[1,2])) >= abs(float(H(0)[1,2]))
then error("09 flow did not suppress S--D mixing")$
if abs(float(Ctot[1,2])) = 0 or determinant(Ctot) <= 0
then error("09 covariance regression failed")$
print("PASS 09 tests: flow invariant, suppression, positive correlated covariance")$
```

Pole sensitivity and local isolation

Listing B.10: Regression test for Model 10

```
batchload("maxima/10_differentiable_scattering.mac")$
for branch in [-1,1] do block([Ep,sa,sf,res,diag_margin],
  Ep:pole(branch,g0), sa:dEdg(Ep,g0),
  sf:(pole(branch,g0+hfd)-pole(branch,g0-hfd))/(2*hfd),
  res:abs(D(Ep,g0)),
  diag_margin:abs(subst(Ep,E,diff(D(E,g0),E)))*radius
              -abs(aa(g0))*radius^2-res,
  if float(res) > 1e-10 then error("10 pole residual"),
  if abs(float(sa-sf)) > 1e-7 then error("10 sensitivity regression"),
  if float(diag_margin) <= 0 then error("10 diagnostic margin"))$
if abs(float(aa(g0))) < 1e-12 or abs(float(bb(g0)^2-4*aa(g0)*cc(g0))) < 1e-12
then error("10 test point is accidentally degenerate")$
if float(abs(pole(1,g0)-pole(-1,g0))) <= 2*radius
then error("10 diagnostic disks overlap")$
print("PASS 10 tests: poles, sensitivities, and floating isolation diagnostics")$
```

Structured-field transversality and safe samples

Listing B.11: Regression test for Model 11

```
batchload("maxima/11_structured_light.mac")$
if float(ka) >= 1 then error("11 target is outside the controlled ka regime")$
for ph in phis do block([kv:kvec(ph),ep:eps(ph),bp],
  bp:cross3(kv,ep)/(c*k0),
  if abs(float(dot3(kv,ep))) > 1e-10 then error("11 E transversality"),
  if abs(float(dot3(kv,bp))) > 1e-10 then error("11 B transversality"),
  if abs(float(dot3(ep,conjugate(ep))-1)) > 1e-10
  then error("11 polarization normalization"))$
for r in samples do
  if abs(float(AE1(r[1],r[2],r[3]))) < 1e-8
  then error("11 sample lies on an E1 node")$
print("PASS 11 tests: ka regime, Maxwell modes, normalization, diagnostics")$
```

Symplecticity, affine composition, and contrast

Listing B.12: Regression test for Model 12

```
batchload("maxima/12_accelerated_quantum_dynamics.mac")$

/* Symplecticity for positive, zero, and negative gradients. */
for Gtest in [Gamma,0,-Gamma] do block([res],
  res:transpose(Sseg(T,Gtest)).J.Sseg(T,Gtest)-J,
  for i thru 2 do for j thru 2 do
    if abs(float(res[i,j])) > 1e-10 then error("12 signed-G symplecticity"))$

/* The entire functions give the exact free-fall map at G=0. */
Szero : Sseg(T,0)$ dzero : dseg(T,g0,0)$
Sfree : matrix([1,T/m],[0,1])$
dfree : matrix([-g0*T^2/2],[-m*g0*T])$
for i thru 2 do for j thru 2 do
  if not is(ratsimp(Szero[i,j]-Sfree[i,j])=0)
  then error("12 exact Gamma=0 symplectic map")$
for i thru 2 do if not is(ratsimp(dzero[i,1]-dfree[i,1])=0)
  then error("12 exact Gamma=0 affine map")$

direct_d : dseg(2*T,g0,Gamma)$
for i thru 2 do
  if abs(float(d2[i,1]-direct_d[i,1])) > 1e-9*(1+abs(float(direct_d[i,1])))
  then error("12 affine composition")$

/* Independent component form of delta^T J Sigma_f J^T delta. */
explicit_quad : Sigmaf[2,2]*delta[1,1]^2
                -2*Sigmaf[1,2]*delta[1,1]*delta[2,1]
                +Sigmaf[1,1]*delta[2,1]^2$
explicit_contrast : exp(-explicit_quad/(2*hbar^2))$
if abs(float(subst(g0,grav,contrast-explicit_contrast))) > 1e-12
then error("12 covariance overlap formula")$
if abs(float(Sigmaf[1,2]-Sigmaf[2,1])) > 1e-30 or determinant(Sigmaf) <= 0
then error("12 propagated covariance")$
if float(subst(g0,grav,contrast)) <= 0 or float(subst(g0,grav,contrast)) > 1
then error("12 contrast interval")$
print("PASS 12 tests: signed/zero gradient, affine map, covariance contrast")$
```

Chapter C

Shell orchestration and book build source

Project layout and executable commands

```
maxima/
|-- README.md
|-- 01_exact_wkb.mac ... 12_accelerated_quantum_dynamics.mac
|-- lib/
|   `-- 02_marchenko_numeric.mac
|-- tests/
|   `-- 01_exact_wkb_test.mac ... 12_accelerated_quantum_dynamics_test.mac
|-- run_models.sh
`-- run_tests.sh
```

Every numbered test batch-loads its matching driver. Model 2 additionally batch-loads `lib/02_marchenko_numeric.mac`. Each shell runner starts every numbered file in a new Maxima process. From the workspace root, the commands are

```
./maxima/run_models.sh
./maxima/run_tests.sh
```

A successful test run ends with one OK or PASS line for each of the twelve topics. The model report preceding each PASS line is output from the batch-loaded driver; the shell exit status is the machine-readable result.

The book build commands and their artifact locations are

```
./Book/book.sh build    # compile Book/book.pdf
./Book/book.sh tests    # save Maxima test output in Book/gar/
./Book/book.sh verify   # tests, build, and artifact-layout verification
./Book/book.sh all      # models, tests, build, and artifact verification
./Book/book.sh clean    # remove auxiliaries while retaining book.pdf
```

The final PDF is `Book/book.pdf`; LaTeX logs and auxiliary files are in `Book/gar/`.

Run all models

Listing C.1: Model runner (`maxima/run_models.sh`)

```
1 #!/usr/bin/env bash
2 set -euo pipefail
3
4 ROOT_DIR="$(CDPATH= cd -- "$(dirname -- "$0")/.." && pwd)"
5 cd "$ROOT_DIR"
6
```

```
for model in maxima/[0-9][0-9]_*.mac; do
  printf '\n=== %s ===\n' "$model"
  maxima --very-quiet --no-init --suppress-input-echo \
    --quit-on-error --batch="$model"
done
```

Run all regression tests

Listing C.2: Test runner (`maxima/run_tests.sh`)

```
#!/usr/bin/env bash
set -euo pipefail

ROOT_DIR="$(CDPATH= cd -- "$(dirname -- "$0")/.." && pwd)"
cd "$ROOT_DIR"

for test_file in maxima/tests/[0-9][0-9]_*.mac; do
  printf '\n=== %s ===\n' "$test_file"
  maxima --very-quiet --no-init --suppress-input-echo \
    --quit-on-error --batch="$test_file"
done
```

Book build script

Listing C.3: LaTeX build source (`Book/book.sh`)

```
#!/usr/bin/env bash
set -euo pipefail

BOOK_DIR="$(CDPATH= cd -- "$(dirname -- "$0")" && pwd)"
PROJECT_DIR="$(CDPATH= cd -- "$BOOK_DIR/.." && pwd)"
AUX_DIR="$BOOK_DIR/gar"
TEX_FILE="$BOOK_DIR/book.tex"
PDF_FILE="$BOOK_DIR/book.pdf"

usage() {
  printf '%s\n' \
    "Usage: ./book.sh [build|models|tests|verify|all|check|clean|distclean]" \
    "" \
    "  build      Compile book.tex and place only book.pdf beside it (default)." \
    "  models     Run all Maxima model drivers; save output under gar/." \
    "  tests      Run all Maxima regression tests; save output under gar/." \
    "  verify     Run tests, build the PDF, and check the artifact layout." \
    "  all        Run models and tests, then build and check the PDF." \
    "  check      Check that book.pdf is readable and auxiliaries did not spill." \
    "  clean      Remove generated files from gar/; keep book.pdf." \
    "  distclean  Run clean and also remove book.pdf."
}

require_command() {
  if ! command -v "$1" >/dev/null 2>&1; then
    printf 'Required command not found: %s\n' "$1" >&2
    exit 127
  fi
}

prepare() {
```

```
  mkdir -p "$AUX_DIR"
}

run_latexmk() {
  latexmk -pdf -norc -interaction=nonstopmode -halt-on-error -file-line-error \
    -outdir="$AUX_DIR" "$TEX_FILE"
}

build_book() {
  require_command latexmk
  require_command biber
  require_command makeindex
  prepare
  cd "$BOOK_DIR"

  # latexmk runs exactly as many passes as the references and contents require.
  # -norc makes the build independent of personal or system latexmk settings.
  run_latexmk

  # BibLaTeX can retain a content-hash rerun request when a required Biber pass
  # produces a byte-for-byte identical .bbl.  A harmless generated-file comment
  # lets the next LaTeX pass confirm that Biber has settled.  Genuine unresolved
  # citations still reappear below and make the build fail.
  if grep -Fq "Please (re)run Biber" "$AUX_DIR/book.log" &&
     [[ -s "$AUX_DIR/book.bbl" ]]; then
    printf '%% BibLaTeX convergence marker; generated by book.sh.\n' \
      >> "$AUX_DIR/book.bbl"
    run_latexmk
  fi

  if grep -Eq 'LaTeX Warning:|Package .+ Warning:|Overfull \\|Underfull \\' \
      "$AUX_DIR/book.log"; then
    printf 'The LaTeX log still contains publication-relevant warnings:\n' >&2
    grep -En 'LaTeX Warning:|Package .+ Warning:|Overfull \\|Underfull \\' \
      "$AUX_DIR/book.log" >&2
    exit 1
  fi

  # Publish only after a complete successful build, leaving all auxiliaries in gar/.
  test -s "$AUX_DIR/book.pdf"
  mv -f "$AUX_DIR/book.pdf" "$PDF_FILE"
  printf 'Built %s\n' "$PDF_FILE"
}

run_models() {
  require_command maxima
  prepare
  cd "$PROJECT_DIR"
  ./maxima/run_models.sh 2>&1 | tee "$AUX_DIR/maxima-models.log"
}

run_tests() {
  require_command maxima
  prepare
  cd "$PROJECT_DIR"
  ./maxima/run_tests.sh 2>&1 | tee "$AUX_DIR/maxima-tests.log"
}

```

```
check_artifacts() {
  require_command pdfinfo
  if [[ ! -s "$PDF_FILE" ]]; then
    printf 'Missing or empty PDF: %s\n' "$PDF_FILE" >&2
    exit 1
  fi

  pdfinfo "$PDF_FILE" | sed -n '1,16p'
  for suffix in aux bbl bcf blg log out toc lof lot lol fls fdb_latexmk run.xml
    synctex.gz; do
    if [[ -e "$BOOK_DIR/book.$suffix" ]]; then
      printf 'Auxiliary file escaped gar/: %s\n' "$BOOK_DIR/book.$suffix" >&2
      exit 1
    fi
  done
  printf 'Artifact layout OK: final PDF in Book/, auxiliaries in Book/gar/.\n'
}

clean_aux() {
  prepare
  rm -f \
    "$AUX_DIR/book.aux" \
    "$AUX_DIR/book.bbl" \
    "$AUX_DIR/book.bbl-SAVE-ERROR" \
    "$AUX_DIR/book.bcf" \
    "$AUX_DIR/book.bcf-SAVE-ERROR" \
    "$AUX_DIR/book.blg" \
    "$AUX_DIR/book-blx.bib" \
    "$AUX_DIR/book.fdb_latexmk" \
    "$AUX_DIR/book.fls" \
    "$AUX_DIR/book.idx" \
    "$AUX_DIR/book.ilg" \
    "$AUX_DIR/book.ind" \
    "$AUX_DIR/book.lof" \
    "$AUX_DIR/book.log" \
    "$AUX_DIR/book.lol" \
    "$AUX_DIR/book.lot" \
    "$AUX_DIR/book.out" \
    "$AUX_DIR/book.pdf" \
    "$AUX_DIR/book.run.xml" \
    "$AUX_DIR/book.synctex.gz" \
    "$AUX_DIR/book.toc" \
    "$AUX_DIR/maxima-models.log" \
    "$AUX_DIR/maxima-tests.log"
  printf 'Cleaned auxiliary files in %s\n' "$AUX_DIR"
}

command_name="${1:-build}"
case "$command_name" in
  build)
    build_book
    ;;
  models)
    run_models
    ;;
  tests)
    run_tests
    ;;
```

```
  verify)
    run_tests
    build_book
    check_artifacts
    ;;
  all)
    run_models
    run_tests
    build_book
    check_artifacts
    ;;
  check)
    check_artifacts
    ;;
  clean)
    clean_aux
    ;;
  distclean)
    clean_aux
    rm -f "$PDF_FILE"
    printf 'Removed %s\n' "$PDF_FILE"
    ;;
  -h|--help|help)
    usage
    ;;
  *)
    printf 'Unknown command: %s\n\n' "$command_name" >&2
    usage >&2
    exit 2
    ;;
esac
```

Index