You Cannot Impose a Boundary Condition on a Volterra Equation

Luminary Quantum Institute, LLC (dba Mitra Institute of Education)

SIRPY · Problem 6 — Unless You Make Room for One

An ongoing series. We are building a public library of problems SIRPY has solved — each entry is one problem, the exact input we handed the solver, the exact output it returned, and an honest read of what happened. No slides. No hand-waving. Numbers you can check.

The obstruction

The previous entry in this library established something that turns out to be inconvenient. A Volterra integral equation **supplies its own initial data**. Set t to the left endpoint and the integral collapses, and the starting value falls out of the equation itself. Nothing is left for the user to specify.

That is elegant, and it creates a problem.

Suppose you want the solution to satisfy an additional condition — say, that it returns to zero at $t = 1$. With a differential equation this is routine: a second-order ODE has two free constants, and boundary conditions are how you spend them. But a Volterra equation has **no free constants**. Its solution is already fully determined. There is nowhere for an extra condition to attach.

Asking for $y(1) = 0$ on top of an already-complete equation is, as posed, over-determined. It should be impossible.

Making room

The resolution is to introduce a degree of freedom deliberately — not in the solution, but in the equation. We leave an unknown constant g in the forcing term:

$$y(t) \;=\; g\,t - t^2 \;+\; \int_0^t (t-s)\,y(s)^2\,ds, \qquad y(1) = 0 \tag{1}$$

Now the problem is well-posed again. The equation still determines y completely — but only *once g is known*. And g is fixed by requiring the resulting solution to satisfy $y(1) = 0$. One unknown constant, one extra condition. The books balance.

This is the same principle that runs through the whole library: name the thing you do not know, and state the condition that determines it.

What we handed the solver

```
#| label: vbc-solve
from sirpy import solve_volterra, plot_verification

r_vbc = solve_volterra(phi="g*t - t**2",   # forcing term with an unknown g
                       kernel="(t-s)",
                       f="y**2",
                       a=0, t1=1,
                       unknown="g",          # name the unknown constant
                       bc="y(1) = 0",        # the condition that fixes it
                       iterations=20, verbose=True)

plot_verification(r_vbc, save="verify_vbc.png", show=True)
```

The unknown is declared with `unknown="g"` and referenced directly in the forcing string. The extra condition is written as `bc="y(1) = 0"` — as it reads. No reformulation, no manually constructed outer loop, no supplied guess.

What came back

The constant, fixed by the terminal condition

```
INFO: the forcing parameter g was fixed by the terminal condition
      y(1) = 0: g = 0.984023080752 (residual 4.72e-16). A Volterra
      equation determines y(0) by itself, so a free parameter in the
      forcing is the only way an extra condition can be imposed.
```

SIRPY found $g = 0.984023080752$, and the terminal condition it was required to satisfy is met to a residual of 4.72×10^{-16} — machine precision.

The second sentence of that message is the mathematics of this problem stated in one line, by the solver, unprompted: because a Volterra equation determines its own starting value, a free parameter in the forcing is the *only* place an additional condition can act.

A structural check that falls out for free

Two further lines are worth reading together.

```
INFO: for this kernel the equation implies y'(0) = g, so the unknown
      'g' is that initial slope.

INFO: the equation supplies its own initial data:
      y(0) = phi(0) = 0,  y'(0) = phi'(0) + lam*K(0,0)*f = 0.984023080752
```

The first is a structural prediction: for this kernel, the equation forces the initial slope to equal g. The second is the initial slope as actually computed from the returned solution — 0.984023080752.

They agree exactly, to every printed digit. The value g arrived at by imposing a condition at the *right* endpoint matches the value the equation independently implies at the *left* endpoint. Nothing required these two to coincide except the solution being correct.

Verified

```
INFO: verification by substitution - VERIFIED: the returned series
      satisfies the Volterra equation to 3.21e-13 (relative 3.21e-13).
```

Table 1: Solver output and verification

Quantity	Value
Recovered constant g	**0.984023080752**
Terminal condition $y(1) = 0$ residual	4.72×10^{-16}
$y'(0)$ implied by the equation	0.984023080752 (matches g)
$y(0)$, determined by the equation	0
Segments covering $[0, 1]$	2 (analytic continuation)
Residual by substitution	3.21×10^{-13}

Honest about branches

One further message deserves quoting in full, because few tools would print it:

```
INFO: no guess= was given, so the shooting started from g = 1. On a
      nonlinear equation the starting value selects a BRANCH: a
      different guess can converge to a different genuine solution,
      and verification will pass on that one too.
```

Read the last clause carefully. SIRPY is telling you that **passing verification does not prove uniqueness**. Another branch may exist, and it would verify just as cleanly, because it too would be a genuine solution.

That is a limitation of the mathematics, not of the solver — and stating it plainly is the difference between a tool that reports and a tool that reassures. We are not claiming this is the only solution. We are claiming this one is real, and showing the residuals that make it checkable.

The figures

What this problem shows

- **An over-determined problem, made well-posed.** A Volterra equation has no free constants; an additional condition needs one created deliberately. SIRPY accepts the unknown, states why it is necessary, and fixes it.
- **The condition is written as it reads.** `unknown="g"` and `bc="y(1) = 0"`. No penalty function, no hand-built outer iteration, no initial guess required.
- **The terminal condition is met to** 4.72×10^{-16} — machine precision on the constraint itself.
- **An independent structural check agrees.** The constant recovered from the right endpoint equals the initial slope the equation implies at the left, to every digit.
- **Uniqueness is not claimed.** The solver states that another branch could exist and would also verify. That disclosure is part of the result.

Next in the series

We continue with problems where the honest, verifiable answer is the hard part — and where having the solution as a *function* rather than a table is what makes certainty possible.

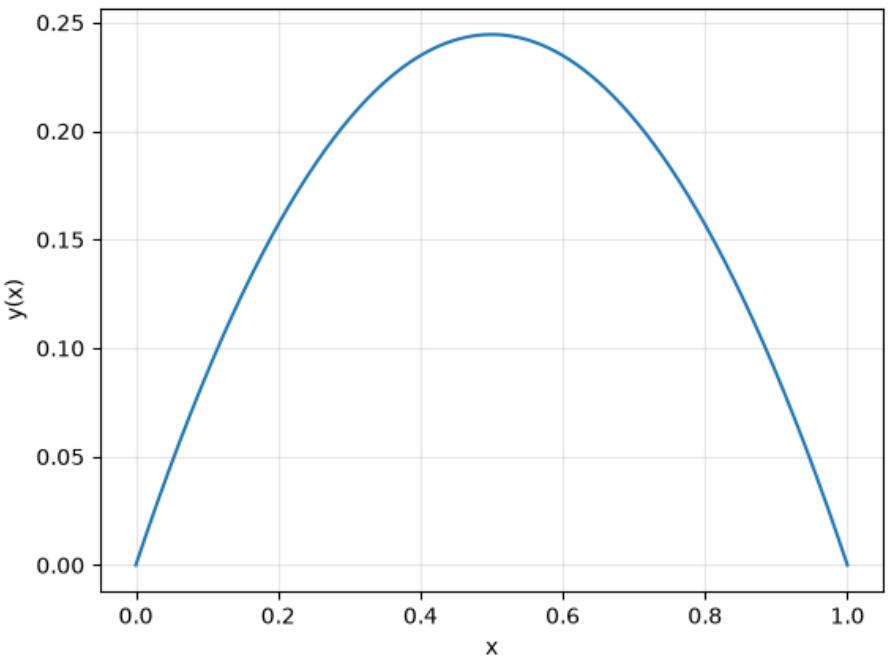

Figure 1: The solution on $[0, 1]$: determined by the equation at the left endpoint, and driven to zero at the right by the recovered constant g.

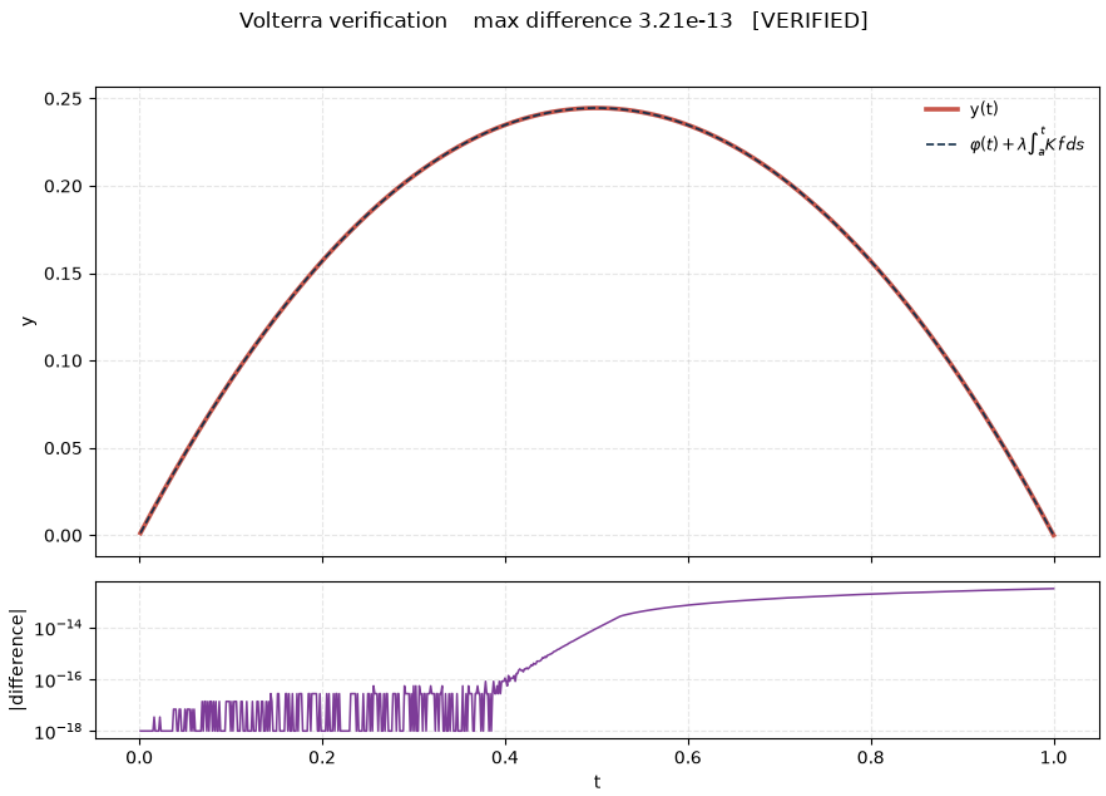

Figure 2: Verification: the returned solution substituted back into the integral equation, with the defect measured across the interval.

The problem library

This entry is part of the SIRPY Problem Library — an ongoing, public catalog of mathematical systems solved and verified by SIRPY.

- **GitHub** — github.com/Mitra-Institute-of-Education-MIE/sirpy-problem-library
- **Web** — mitrainstituteofeducation.org/sirpy

Submit a problem

Have a differential or integral equation you believe is challenging, unusual, or simply beautiful? We would like to see it. We will run it through SIRPY and report honestly what happened — if it solves it, how; if it struggles, why; if it reaches a current limit, we will say so.

Contact: info@mitrainstituteofeducation.org · mitrainstituteofeducation.org/contact

Please only send problems you are free to share publicly.

SIRPY is in final validation prior to its first public release. Every line quoted above is verbatim from a single run of the solver; nothing is reproduced from memory.