Rope around the earth

by *R. Wm. Gosper*

(This article is about series reversion, especially as a "poor man's TAYLOR_SOLVE".)

Remember the old chestnut about how much slack would you need to raise a rope encircling equator by three feet? If instead, you take up all that slack by hoisting at one point, the question of how high is suddenly nontrivial.

Let h be the unknown height, r the radius of the earth, and $2\pi\epsilon$ the slack (permitting an all around elevation of ϵ, *e.g.,* 3ft.). Let α be the angle at the center of the earth between the hoist point and the rope's tangency at the "horizon". Then the equations are

```
(C1) r/(r+h) = cos(α);
```

$$\text{(D1)}\qquad \frac{R}{R+H} = \cos(\alpha)$$

```
(C2) 2*(r*tan(α)+r*(π-α)) = 2*π*(r+ϵ);
```

$$\text{(D2)}\qquad 2\,(\tan(\alpha)\,R + (\pi - \alpha)\,R) = 2\,\pi\,(R + \epsilon)$$

Eliminating α,

```
(C3) subst(solve(d1,α)[1],%);
SOLVE is using arc-trig functions to get a solution.
Some solutions may be lost.
```

$$\text{(D3)}\qquad 2\left(\sqrt{\left(\frac{H}{R}+1\right)^2-1}\,R + \left(\pi - \operatorname{asec}\left(\frac{H}{R}+1\right)\right)R\right) = 2\,\pi\,(R+\epsilon)$$

Now we wish to solve for h in terms of ϵ. But h is inside both a square root and an arcsecant, so we have no hope of $h(\epsilon)$ in closed form, and must settle for a series approximation. An easy way to do this is to revert the series for the readily available $\epsilon(h)$:

```
(C4) solve(%,ϵ)[1];
```

$$\text{(D4)}\qquad \epsilon = \frac{\sqrt{\left(\frac{H}{R}+1\right)^2-1}\,R - \operatorname{asec}\left(\frac{H}{R}+1\right)R}{\pi}$$

```
(C5) block([taylor_simplifier:lambda([x],rootscontract(radcan(x)))], taylor_revert(%,h,2));
```

$$\text{(D5)/T/}\qquad H = \frac{(9\,\pi^2\,R)^{1/3}\,\epsilon^{2/3}}{2} + \frac{(9\,(\frac{3\,\pi^4}{R})^{1/3})\,\epsilon^{4/3}}{40} - \frac{(9\,\pi^2)^2\,\epsilon^2}{2800\,R} + \ldots$$

(C5) here exhibits two new features: the TAYLOR_SIMPLIFIER may now be a LAMBDA expression, and TAYLOR_REVERT has gotten a lot smarter. We shall see how smart presently, but first, let's play with its answer. Plugging in for earth's radius and the rope height (in feet),

```
(C6) block([boxchar:" "],h=sfloat(subst([r=3959*5280,ϵ=3],map('box,rhs(%)))));
```

$$\text{(D6)}\qquad H = 1278.3162 + 0.023451889 + -1.3658636\text{e-}8$$

I.e., the Empire State Building + a quarter inch − four nanometers. (Are you still disappointed we couldn't

get an exact solution?) This peculiar height, neither earthlike nor footlike, comes from the unusual form of the leading term: a geometric mean of ϵ, ϵ and r. (Preserving dimensions of distance, as must all the terms.) As a check, we can plug this series approximation for h into the equation for ϵ.

```
(C7) block([taylor_simplifier:radcan],subst(d5,d4));
```

(D7)/T/ $\epsilon = \epsilon + . \quad . \quad .$

With the default `TAYLOR_SIMPLIFIER`, this `SUBST` produces superfluous terms which are all unsimplified 0s. Then postnatal `RADCAN` gives $\epsilon = \epsilon$.

Let's also check our numerical result.

```
(C8) sfloat(subst([h=1278.34,r=3959*5280],d4));
```

(D8) $\epsilon = -1.4996331$

Wha? A foot and a half underground? This was supposed to be good to nanometers!

```
(C9) sfloat(subst([h=1278.34d0,r=3959*5280],d4));
```

(D9) $\epsilon = 3.0000036$

Whew! The problem was the ferocious numerical instability of $\operatorname{arcsec}(1 + h/r)$ for very small h/r. Adding the 1 killed much of the accuracy, while the near vertical slope of arcsec near 1 and subtractive cancellation killed the rest. Note that it was not necessary to supply a very accurate value of h, but it was necessary that it be double precision (even though it was computed with single!), so that both occurrences in `(D4)` are consistent at the time of subtraction. (Yet people still regard numeric arithmetic routine and symbolic arithmetic exotic.)

A point of style: instead of `h=1278.34d0`, I should have typed `dfloat(rembox(d6))`, for greater accuracy and lesser likelihood of meatware screwup. This, in combination with `dfloat`s instead of `sfloat`s in `(C5)` and `(C9)`, would have recovered our nanometers.

`TAYLOR_REVERT` takes as its first argument a series, expression, or equation to be reverted. The second argument, if present and non-numeric, is the variable to be (approximately) "solved for". If absent, it is guessed, *e.g.*, from the `TAYLORINFO` of the first argument. Instead of the variable, you can even supply an equation of the form variable = point, to specify the pre-reversion expansion point. Otherwise, this point is deduced from `TAYLORINFO`, or defaulted to 0, as in the old days. A numeric second or third argument determines the order of expansion of the solution, when reverting an equation, or inverse function when reverting a simple expression or series. *I.e.*, `TAYLOR_REVERT` figures out for you how many input terms it needs for your specified output degree.

However, if your "revertand" is already expanded in a series, and you neglect to specify an output degree, `TAYLOR_REVERT` will automatically find the maximum number of valid terms, in keeping with the usual behavior of `TAYLOR` expansions:

```
(C10) block([taylor_simplifier:'radcan],taylor_revert(d5));
```

(D10)/T/

$$\epsilon = \frac{(2\,\mathrm{SQRT}(R)\,\mathrm{SQRT}(2))\,H^{3/2}}{3\,R\,\pi} - \frac{(3\,\mathrm{SQRT}(R)\,\mathrm{SQRT}(2))\,H^{5/2}}{10\,R^2\,\pi} + \frac{(23\,\mathrm{SQRT}(R)\,\mathrm{SQRT}(2))\,H^{7/2}}{112\,R^3\,\pi} + . \quad . \quad .$$

```
(C11) d4-%;
```

(D11)/T/ $0 = 0 + . \quad . \quad .$

(D10) shows that the TAYLOR_REVERT in (C5) had to expand to $h^{7/2}$ to reach the requested accuracy of ϵ^2.

Let's do a simple quintic.

```
(C12) y = x^5-x;
                                          5
(D12)                                Y = X  - X
(C13) taylor_revert(%,15);
                                    5      9       13
(D13)/T/                 X = - Y - Y  - 5 Y  - 35 Y   + .  .  .
(C14) %^5-%;
                                   5
(D14)/T/                          X  - X = Y + .  .  .
(C15) taylor_revert(d12,x=1,6);
                                 2      3        4    5          6
                          Y   5 Y    5 Y    385 Y    Y    23205 Y
(D15)/T/        X = 1 + - - ---- + ---- - ------ + -- - -------- + .  .  .
                          4    32     32     2048    4     65536
(C16) %^5-%;
                                   5
(D16)/T/                          X  - X = Y + .  .  .
```

Notice that we found a different root (for small y) by expanding at $x = 1$ instead of the default 0. You can think of the expansion point of the supplied equation as a first approximation (constant term) of the series solution.

Hoary and regrettable mathematical tradition defines series reversion to produce a series (rather than an equation) for the inverse function in terms of the *same variable* as the given series (again, not an equation). Hoariness is still the default:

```
(C17) taylor(%e^x,x,0,4);
                                     2    3    4
                                    X    X    X
(D17)/T/                  1 + X + -- + -- + -- + .  .  .
                                    2    6    24
(C18) taylor_revert(%);
                                 2          3          4
                          (X - 1)    (X - 1)    (X - 1)
(D18)/T/          X - 1 - -------- + -------- - -------- + .  .  .
                             2          3          4
```

Without resorting to equations, we can partially alleviate this symbol confusion by mentioning a second, new variable as the third or fourth argument:

```
(C19) taylor_revert(%,x,y);
                                     2    3    4
                                    Y    Y    Y
(D19)/T/                  1 + Y + -- + -- + -- + .  .  .
                                    2    6    24
```

This might be useful in programs, but interactively, we recommend the equation form, `taylor_revert(y=d18)`, giving `X = 1 + Y +` *etc.*, as the best way to avoid mistaking functions for their inverses. This gives us, in effect, a poor man's TAYLOR_SOLVE. The real TAYLOR_SOLVE is considerably more ambitious, but in many ways, less convenient. Its arguments are, respectively: equation, unknown, expansion variable, expansion point, and expansion degree. It finds many solutions that are missed by simple reversion (see (C34),(D34)), but, in its zeal, sometimes gets lost. Also, you may prefer to guess (for TAYLOR_REVERT) the *input* expansion

point, in effect, a first approximation to the solution, rather than to specify a good place for TAYLOR_SOLVE to expand its result.

Here are two derivations of the real fixed points of tan.

```
(C20) x=tan(x);
(D20)                                  X = TAN(X)
```

A look at the graph tells us that $y = x$ intersects $y = \tan(x)$ just short of midway between integer multiples of π.

```
(C21) subst(x=π*(integer+1/2)-ϵ,%);
                                        1
(D21)                  π (INTEGER + -) - ϵ = COT(ϵ)
                                        2

(C22) taylor_revert(%,ϵ,0,integer,5);

                  1                    2                 13
(D22)/T/ ϵ = --------------- + --------------------- + ----------------------
                      1          3              1 3          5              1 5
           π (INTEGER + -)   (3 π )  (INTEGER + -)    (15 π )  (INTEGER + -)
                      2                         2                        2

                                                                    + . . .

(C23) subst(%,first(c21));

                      1          1                  2
(D23)/T/ X = π (INTEGER + -) - --------------- - ---------------------
                      2                1              3              1 3
                           π (INTEGER + -)   (3 π )  (INTEGER + -)
                                      2                         2

                                                  13
                                          - ---------------------- + . . .
                                                 5              1 5
                                            (15 π )  (INTEGER + -)
                                                               2

(C24) %-map('tan,%);

                                 146                  19
(D24)/T/ X - TAN(X) = - ------------------------ + ---------------------- + . . .
                              5              1 5          7              1 7
                        (105 π )  (INTEGER + -)    (63 π )  (INTEGER + -)
                                           2                         2
```

The fifth order term here is actually correct, and does not indicate a problem with the fifth order term of (D23). However, Macsyma should have discarded the seventh order term. It's on our list.

A more direct approach:

```
(C25) x=atan(x)+integer*π;
(D25)                       X = ATAN(X) + π INTEGER

(C26) taylor_revert(solve(%,integer)[1],x=∞,5);

                      1          1                  2
(D26)/T/  X = π (INTEGER + -) - --------------- - ---------------------
                      2                1              3              1 3
                           π (INTEGER + -)   (3 π )  (INTEGER + -)
                                      2                         2

                                                  13
                                          - ---------------------- + . . .
                                                 5              1 5
                                            (15 π )  (INTEGER + -)
                                                               2
```

What about cases like

```
(C27) x^5+y^5=x*y;

                                        5    5
(D27)                                  Y  + X  = X Y
```

where you can't explicitly solve for either variable? Solve for 0!

```
(C28) zero=lhs(%)-rhs(%);

                                        5          5
(D28)                             ZERO = Y  - X Y + X

(C29) taylor_revert(%,x,9);

                                 5              5 5              5 9
                        ZERO - Y     (ZERO - Y )    5 (ZERO - Y )
(D29)/T/           X = - --------- - ------------ - -------------- + .  .  .
                            Y               6                11
                                           Y                Y

(C30) taylor(subst(0,zero,%),y,0,5*9-11);

                                     4    19      34
(D30)/T/                   X + . . . = Y  + Y   + 5 Y   + . . .

(C31) y^5+%^5-%*y;

                             5           5
(D31)/T/                    X  - X Y + Y + . . . = 0 + . . .
```

Of course, this is a little crazy, and only after seeing (D29) would you likely have a clue as to the degree of the approximation. (If negative, (C30) could try expanding at ∞!) Also, when things get this theological, forget not the teachings of Newton, the Methodist:

```
(C32) (rhs(d28),factor(x-%%/diff(%%,x)));

                                         5      5
                                        Y  - 4 X
(D32)                                   ---------
                                                4
                                        Y - 5 X

(C33) x=subst(rhs(d30),x,%);

                                4    19      34       49
(D33)/T/                   X = Y  + Y   + 5 Y   + 35 Y   + .  .  .
```

Comparing the coefficients in (D33) with those of (D13), we kick ourselves for failing to notice that `sublis([x=-x*y^(1/15),y=y^(4/15)],d27)` gives (D12), a problem we already solved. (In fact, counting from $n = 0$, the n^{th} coefficient is $\binom{5n}{n}/(4n+1)$.)

In fairness to TAYLOR_SOLVE,

```
(C34) taylor_solve(x^5+y^5 = x*y,x,y,0,22);

                                                   4        3   31/4         2   23/2
                     4    19                 1/4  Y    (5 K0 ) Y        (5 K0 ) Y
(D34)/T/  [[X = Y  + Y   + . . .], [X = K0 Y    - -- - ------------- - -------------
                                                  4         32               32

                                                       61/4     19
                                              (385 K0) Y       Y           4
                                            - -------------- - --- + . . ., K0  = 1]]
                                                   2048         4
```

Simpleminded reversion completely overlooked the second four solutions, although we can get the K0 = 1

case by reverting (D30), or more properly, the solution of (D28) for $y(x)$ instead of $x(y)$.

Closing sermonette: you (along with various math software vendors) have perhaps perceived an analogy between arithmetic on series approximations and arithmetic on variable precision floating point numbers. This analogy is false! For example, notice that it was possible for Macsyma to deduce an additional term in (D33), whereas if X and Y were numbers which attempted to carry with them estimates of their own (im)precision, there would be no convergence. At best, the "precision" would remain stuck at whatever was advertised by the first, presumably crude, approximation.

This is why Macsyma performs its "bigfloat" operations according to an externally imposed precision, rather than attempting to assign precisions to its numerical operands. Numbers, unlike physical measurements, are not uncertain. It is the operations on them that determine the uncertainty. "Gee, Mr. Interval Arithmetic, how much is $x - x$?"

This question raises the possibility that symbolic analysis of a whole expression could alert a clever evaluator to boost (or relinquish) the numerical precision of individual arithmetic steps. Indeed, a symbolic version might have alerted the TAYLOR mechanism to go for the y^{64} term in (D33). But, again, precision is determined by the operation (in this case, compound), rather than the operands. Even when some futuristic error analyzer extends from expressions to whole algorithms, numerical precision will be imposed from without. Until then, beware of letting the numbers try to figure it out for themselves.

P.S.: The figure (except for subsequent lettering) was made by

$$\text{PILINE(LOX, HIX)} := \text{LOX} + \frac{(\text{HIX} - \text{LOX})\ (\text{T} + \pi)}{2\ \pi}$$

```
block([equalscale:true,plotnum:69],apply('paramplot,
 sfloat(subst([β=π/2-acos(lhs(d1)),totaldisrep(taylor_revert(d4,10/3)),r=1,ϵ=.1],
 [[r*cos(t)-2.5+ϵ,(r+ϵ)*cos(t)-2.5+ϵ,r*cos(t),r*cos(β)*piline(-1,1),r*cos(β)*piline(0,1)],
  [r*sin(t),      (r+ϵ)*sin(t),      r*sin(t),r+h-abs((h+r*(1-sin(β)))*piline(-1,1)),
                                                                    r*sin(β)*piline(0,1)],
  t,-π,π]])))$
```