# Floating-Point Tricks to Solve Boundary-Value Problems Faster

Prof. W. Kahan
Math. and Computer Sci. Depts.
Univ. of Calif. @ Berkeley

**§0. Abstract:** Old tricks are exhumed to accelerate the numerical solution of certain discretized boundary-value problems. Without the tricks, half the digits carried by the arithmetic can be lost to roundoff when the discretization's grid-gaps get very small. The tricks can procure adequate accuracy from arithmetic with `float` variables 4-bytes wide instead of `double` variables 8-bytes wide that move slower through the computer's memory system and pipelines. The tricks are tricky for programs written in MATLAB™ 7+, JAVA, FORTRAN and post-1985 ANSI ***C***. For the original Kernighan-Ritchie ***C*** of the 1970s, and for the few implementations of ***C***99 that fully support IEEE Standard 754 for Binary Floating-Point, most of the tricks are easy or unnecessary. Their efficacy is illustrated here by examples.

**Contents:**



Posted at `www.eecs.berkeley.edu/~wkahan/Math128/FloTrik.pdf`

**§1. Introduction:** Computations, formerly carried out in 8-byte-wide `double` floating-point, could afford to lose over half the arithmetic's 16 sig.dec. and yet retain accuracy adequate for almost every requirement by scientists and engineers. Now they are tempted to replace `double` by 4-byte-wide `float` variables that will move twice as fast through computers' pipelines and vast memories, dissipate half the energy, and take advantage of inexpensive hardware mass-produced for entertainment and communications. But this replacement exacerbates the threat to accuracy from roundoff that was formerly ignored. Only if unnoticed can the loss of about half the 7 sig.dec. of `float` arithmetic be ignored.

There are tricks that defend discretized differential equations against excessive loss to roundoff.

A few of us used these tricks in the 1970s during the brief reign of small computers with `float` arithmetic hardware but not `double`. The tricks ran faster than `double` simulated in software. The tricks relied upon a property of floating-point subtractive cancellation:

If $p$ and $q$ are floating-point numbers of the same precision,
and if $1/2 \le p/q \le 2$ ,
then $p - q$ is computed exactly, unsullied by a rounding error.

A proof appeared on p. 138 of *Floating-Point Computation* by P.H. Sterbenz (1974, Prentice-Hall, NJ). Rare exceptions occurred on perverse hardware lacking a g*uard digit*, and for abrupt instead of *gradual* underflow of $p - q$ ; these exceptions do not happen nowadays on hardware conforming fully to IEEE Standard 754.

The tricks entail complexities that bloat a computer program's capture-cross-section for mistakes.

No endorsement of these tricks is implied by their lengthy discussions and analyses below. Quite the contrary. Their complexities are a penalty imposed by programming languages and compilers lacking convenient support for arithmetic operations more precise than their operands. A major exception was ***C*** designed by B.W. Kernighan and D.M. Ritchie for an early DEC PDP-11. Its floating-point board required a call upon the operating system to select one of `float` or `double` precision. For speed's sake, the board was left in `double`; consequently every floating-point expression was evaluated in `double` regardless of whether operands were `floats`. This practice was numerically advantageous. It rendered unnecessary almost all the tricks exhibited below, and greatly enhanced the accuracy and reliability of many a FORTRAN program transliterated into ***C*** , especially 3-dimensional geometrical computations like those described on my web page's `www.eecs.berkeley.edu/~wkahan/MathH110/Cross.pdf` . But in the mid 1980s, before those advantages were appreciated widely, ANSI committee X3J11 let ***C*** compilers revert to *Fortranish* expression-evaluation. Most did. Now tricks are necessary for most ***C*** compilers.

The simplest trick is *Compensated Summation* used to suppress the worst rounding errors in the numerical solution of initial-value problems to which one-dimensional boundary-value problems are converted when solved by *Shooting* methods. These deliver better accuracy as a *stepsize* $\theta$ is diminished, though at the cost of greater work proportional to $1/\theta$ . Without that trick or else extra-precise arithmetic, the enhanced accuracy is vitiated by rounding errors that accumulate proportional to $1/\theta$ in worst cases, though these happen only rarely. An example of damaging accumulation is provided by an initial-value problem in §§2-3.

The solution $u(\mathbf{x})$ of a boundary-value problem $\mathrm{div}(\boldsymbol{p}\cdot\mathbf{grad}\,u) + q\cdot u = r$ is often a potential computed only to permit the subsequent computation of a vector force-field $\mathbf{grad}\,u$ from finite-difference formulas. Because these formulas amplify errors in $u$ it must be computed accurately enough that subsequent subtractive cancellations will not leave too few correct digits to determine $\mathbf{grad}\,u$ as accurately as it is needed. This accuracy is gained by computing $u$ over a sufficiently refined grid of mesh-points. As happens in §9, mesh refinement can worsen the contamination of $u$ by roundoff unless the program acts to abate that contamination. The simplest abatement by far resorts to extra-precise arithmetic. When this is unavailable or too slow, the abatement must use a tricky trick presented in these notes in §7. An example in §8 will test in §10 how well the trick works to compute the regular solution of a singular boundary-value problem.

After it is explained for a second-order *ordinary* differential equation $(P\cdot U')' + Q\cdot U = R$ with boundary conditions at the ends of some interval, the trick will be applied to an elliptic *partial* differential equation on a squaare in §??. Further elaborating the trick to work for parabolic and hyperbolic partial differential equations that characterize propagation may incur so much extra memory traffic as to vitiate the trick; but that's a story for another day.

**§2. Discretized Initial-Value Problem:** A numerical solution $\mathbf{Y}(\tau)$ of the differential equation

$$d\mathbf{y}/d\tau = \mathbf{f}(\mathbf{y}) \quad \text{over a given interval} \quad 0 \le \tau \le T \quad \text{with a given} \quad \mathbf{y}(0) := \mathbf{y}^\circ$$

is to be computed either as the terminal $\mathbf{Y}(T)$ or as $\mathbf{Y}(\tau)$ to be plotted over that interval. Most numerical methods resemble the conversion of the differential equation into an integral equation

$$\mathbf{y}(\tau+\theta) = \mathbf{y}(\tau) + \int_0^\theta \mathbf{f}(\mathbf{y}(\tau+\sigma))\cdot d\sigma\,,$$

because an approximation $\mathbf{Y}(\tau) \approx \mathbf{y}(\tau)$ is updated repeatedly, for $\tau = 0, \theta, 2\theta, 3\theta, \ldots, T-\theta$, to

$$\mathbf{Y}(\tau+\theta) := \mathbf{Y}(\tau) + \mathbf{F}(\mathbf{Y}(\ldots), \theta)\cdot\theta$$

wherein $\mathbf{F}(\mathbf{Y}(\ldots), \theta)$ extracts samples of $\mathbf{f}(\mathbf{Y}(\ldots))$ to estimate the average $\int_0^\theta \mathbf{f}(\mathbf{y}(\tau+\sigma))\cdot d\sigma/\theta$. (The stepsize $\theta$ may vary with $\tau$ but has been kept constant here to simplify the exposition.)

Absent roundoff, the error $\mathbf{Y}(T) - \mathbf{y}(T) \to \mathbf{o}$ like $\theta^{Order}$; the exponent *Order* depends upon the details of $\mathbf{F}(\ldots)$ and always exceeds 1, often exceeds 3. When adequate accuracy can be achieved only by choosing a very tiny $\theta$, rounding errors interfere. The worst of them occur at the additions of $\mathbf{Y} + \mathbf{F}\cdot\theta$. Here is how roundoff loses digits:

```
YYYYYYY          = Y(τ)
+  FFFFfff·θ     + F·θ   as if   ffffffff
-------                            lost
YYYYYYY          = Y(τ+θ)
```

Thus rounding errors appear to inject uncertainty proportional to $\varepsilon/\theta$ into $\mathbf{F}$ (and hence into $\mathbf{f}$); $\varepsilon$ is the arithmetic's roundoff threshold; `float`'s $\varepsilon = 2^{-23} \approx$ 1e-7; `double`'s $\varepsilon = 2^{-52} \approx$ 2e-16. That injection limits the accuracy achievable in $\mathbf{Y}$, as if roundoff and discretization conspired to lose at least some fraction like $1/(1 + Order)$ of the arithmetic's digits of $\mathbf{f}$. That loss might often have gone unnoticed when the arithmetic was `double`. Not so likely if `float`.

A palliative is the choice of a higher *Order* formula for **F**. It works only if the solution **y**(τ) is smooth enough, and the needed accuracy high enough, that the higher *Order* formula allows a substantially bigger stepsize θ, whence substantially fewer updating steps and fewer lost digits.

A remedy is the use of extra-precise arithmetic, storing **Y** to at least several more sig.bits than are trusted in **f** or desired in **y**. When this remedy is too slow or unavailable, the only remedy is *Compensated Summation*:

```
Y := y° ;  cY := o ;                          … Initialize Y and its compensatory cY
For τ = 0 to T–θ in steps of θ {
      Yo := Y ;
      ΔY := cY + F(Y(…), θ)·θ ;
      Y := Yo + ΔY ;                          … rounded, losing digits FFF·θ
      cY := (Yo – Y) + ΔY ; }                 … recovers them. (HONOR PARENTHESES!)
```

This trick is inefficient when the differential equation is so stable that it forgets its earlier errors, or so unstable that the propagated growth of the earliest few errors overwhelms all later errors.

**§3. An Example:** The chosen differential equation $d\mathbf{y}/d\tau = \mathbf{f}(\mathbf{y})$ has terminal $T := 65/32$ and

$$\mathbf{y} := \begin{bmatrix} v \\ w \\ \tau \end{bmatrix}, \quad \mathbf{f}(\mathbf{y}) := \begin{bmatrix} w/\tau \\ -4\tau\cdot(1-\tau)\cdot(1+\tau)\cdot v \\ 1 \end{bmatrix} \quad \text{and} \quad \mathbf{y}(0) = \mathbf{y}^\circ := \begin{bmatrix} 2^{29} \\ 0 \\ 0 \end{bmatrix}.$$

This singular differential equation has a regular solution $v(\tau) = 2^{29} e^{-\tau^2}$, $w(\tau) = -2\tau^2\cdot v(\tau)$. The singularity was removed numerically by substituting "$w/(\tau+\eta)$" for "$w/\tau$" in **f**, where η barely exceeds the underflow threshold, thus replacing an invalid 0/0 operation by 0 with no other effect upon the computation of **f**, because "$w/(\tau+\eta)$" rounds to "$w/\tau$" if $\tau \neq 0$.

The chosen numerical method is a classical 4th-order Runge-Kutta formula whose **F**·θ is …

$$\mathbf{F}(\mathbf{Y}(\dots), \theta)\cdot\theta = \left( 2\cdot(h\mathbf{F}_1 + h\mathbf{F}_3) + 4\cdot h\mathbf{F}_2 + h\mathbf{F}_4 \right)/6 \text{ wherein}$$

$$h\mathbf{F}_1 := \frac{\theta}{2}\cdot\mathbf{f}(\mathbf{Y}) ; \quad h\mathbf{F}_2 := \frac{\theta}{2}\cdot\mathbf{f}(\mathbf{Y} + h\mathbf{F}_1) ; \quad h\mathbf{F}_3 := \theta\cdot\mathbf{f}(\mathbf{Y} + h\mathbf{F}_2) ; \quad h\mathbf{F}_4 := \theta\cdot\mathbf{f}(\mathbf{Y} + h\mathbf{F}_3) ;$$

The chosen number $n := 2560$ of steps produced a stepsize $\theta = T/n$ *exactly*. All arithmetic and variables were 24-sig.bit `float`. Computed results for **Y**(T)'s first component were …

| | |
|---|---|
| V(T) = 867**0448** | computed without Compensated Summation |
| V(T) = 866924**1** | computed with Compensated Summation |
| v(T) ≈ 8669240 | the true v(T) rounded to 24 sig,bits. |

Compensated Summation has reduced this example's loss of accuracy in **Y**(T) from over 10 sig.bits to less than 2 of the arithmetic's 24.

This example also reminds us that no simple foolproof way exists to infer **Y**(T)'s error from the accuracy of the compensated updating formula $\mathbf{Y}(\tau+\theta) := (c\mathbf{Y} + \mathbf{F}(\mathbf{Y}(\dots), \theta)\cdot\theta) + \mathbf{Y}(\tau)$. A simple way repeats the computation of **Y**(T) with a sequence of diminishing stepsizes θ until as many digits of **Y**(T) converge to presumed digits of **y**(T) as roundoff (in **f**, θ and T too) allows.

**§4. Discretized Boundary-Value Problem:** Suppose $P$, $Q$ and $R$ are scalar-valued functions of the scalar independent variable $x$, and $Q$ and $R$ may depend also upon the scalar solution $U(x)$ of the differential equation $(P{\cdot}U')' + Q{\cdot}U = R$. We assume that $P$, $Q$ and $R$ are smooth functions to preclude distracting complications. Choose a sequence $x_0 < x_1 < x_2 < \ldots < x_N$ of mesh-points to span the interval over which the solution $U(x)$ is to be computed; they can be spaced non-uniformly so long as every gap $h_j := x_{j+1} - x_j$ is small. Let $\mathbf{u}_j \approx U(x_j)$ numerically and then set, say, $p_{j+1/2} := P((x_j+x_{j+1})/2)$, $\mathbf{q}_j := Q(x_j, u_j)$ and $\mathbf{r}_j := R(x_j, \mathbf{u}_j)$. One of several discretized approximations to the derivative $(P{\cdot}U')'$ at $x = x_j$ is the difference-quotient

$$2{\cdot}\big( p_{j+1/2}{\cdot}(\mathbf{u}_{j+1} - \mathbf{u}_j)/h_j - p_{j-1/2}{\cdot}(\mathbf{u}_j - \mathbf{u}_{j-1})/h_{j-1} \big)/(h_j + h_{j-1}) = (P{\cdot}U')' + O(|h_j - h_{j-1}| + (h_j + h_{j-1})^2) .$$

(Eliminating the term $|h_j - h_{j-1}|$ complicates the exposition without affecting the trick; see Appendix 2 below.)

Substituting this approximation into the differential equation $(P{\cdot}U')' + Q{\cdot}U = R$ at every mesh-point produces an (almost) linear system $(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \mathbf{r}$ of equations in which $\mathbf{u}$ is a column of unknowns $\mathbf{u}_j$, $\mathrm{Diag}(\mathbf{q})$ is a diagonal matrix computed from the elements $\mathbf{q}_j$ and gaps $h_j$, column $\mathbf{r}$ is computed from the elements $\mathbf{r}_j$ and gaps $h_j$, and $\mathbb{T}$ is a tridiagonal matrix computed from the elements $p_{j+1/2}$ and gaps $h_j$. The bottom and topmost entries in $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ and $\mathbf{r}$ include contributions from the boundary-value problem's boundary conditions.

Sometimes $\mathrm{Diag}(\mathbf{q})$ is supplanted by a tridiagonal matrix to help approximate the differential equation better. For the same reason $\mathbb{T}$ may become five-diagonal; but we shall disregard these possibilities in what follows since they can be accommodated by a straightforward elaboration of a trick whose description we still hope to keep simple.

The equation $(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \mathbf{r}$ has to be solved for the desired $\mathbf{u} = (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))^{-1}\cdot\mathbf{r}$. Even if $\mathbf{q}$ and $\mathbf{r}$ are independent of $\mathbf{u}$, the solution process will usually require iteration if only to attenuate obscuration by roundoff during the solution process. One process, akin to Gaussian elimination, factorizes $\mathbb{T} + \mathrm{Diag}(\mathbf{q}) \approx \mathrm{E}\cdot\mathrm{B}$ wherein $\mathrm{B}$ is bidiagonal and upper-triangular, and $\mathrm{E}$ is a bidiagonal lower-triangular matrix or else one whose rows have been permuted by pivoting during the factorization process. These factors serve to compute $\mathbf{u} \approx \mathrm{B}^{-1}\cdot(\mathrm{E}^{-1}\cdot\mathbf{r})$ by first forward substitution (perhaps permuted) to compute $\mathrm{E}^{-1}\cdot\mathbf{r}$ and then back-substitution to get $\mathbf{u}$.

If $\mathbf{q}$ and $\mathbf{r}$ depend upon $\mathbf{u}$ they will have been estimated from a guess at $\mathbf{u}$ and must now be recomputed from the latest estimate of $\mathbf{u}$, after which their changes must be taken into account during the computation of an improved estimate $\mathbf{u} + \Delta\mathbf{u}$ to supplant the one just computed. How this is done depends upon how strongly $\mathbf{q}$ and $\mathbf{r}$ depend upon $\mathbf{u}$:

- If $\mathbf{q}$ and $\mathbf{r}$ depend very weakly upon $\mathbf{u}$ then $\Delta\mathbf{u} \approx \mathrm{B}^{-1}\cdot\big(\mathrm{E}^{-1}\cdot\big(\mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}\big)\big)$ using the latest values for $\mathbf{u}$, $\mathbf{q}$ and $\mathbf{r}$ but the same factors $\mathrm{E}$ and $\mathrm{B}$ as before.
- If $\mathbf{q}$ and $\mathbf{r}$ depend weakly but not very upon $\mathbf{u}$ then $\Delta\mathbf{u} \approx \mathrm{B}^{-1}\cdot\big(\mathrm{E}^{-1}\cdot\big(\mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}\big)\big)$ using the latest values $\mathbf{u}$, $\mathbf{q}$ and $\mathbf{r}$ and a recomputed factorization $\mathbb{T} + \mathrm{Diag}(\mathbf{q}) \approx \mathrm{E}\cdot\mathrm{B}$.
- If $\mathbf{q}$ and $\mathbf{r}$ depend strongly upon $\mathbf{u}$ then $\Delta\mathbf{u} \approx \mathrm{B}^{-1}\cdot\big(\mathrm{E}^{-1}\cdot\big(\mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}\big)\big)$ using the latest values for $\mathbf{u}$, $\mathbf{q}$ and $\mathbf{r}$ and recomputed factors $\mathrm{E}$ and $\mathrm{B}$ that take the columns $\partial\mathbf{q}/\partial\mathbf{u}$ and $\partial\mathbf{r}/\partial\mathbf{u}$ into account. This complicates the process but does not affect the trick.

The foregoing process has been called "Iterative Refinement" among other things. Ideally, at most a few iterations $\mathbf{u} \to \mathbf{u} + \Delta\mathbf{u}$ should suffice to solve the equation $(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \mathbf{r}$ for $\mathbf{u}$ as accurately as the data $\mathbb{T}$, $\mathbf{q}$ and $\mathbf{r}$ deserve. No matter how the equation's solution $\mathbf{u}$ is computed, its accuracy turns out to be limited mostly by the accuracies of successive residuals $\mathbf{s} := \mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$. The trick is to compute each $\mathbf{s}$ accurately enough, as we shall see.

Usually scalar factors dependent only upon the gaps $h_j$ have been incorporated into the rows of the expression "$\mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$" so that it can be computed repeatedly for different columns $\mathbf{u}$ without incurring repeated divisions by expressions dependent only upon the gaps. Sometimes these scalar factors fail to keep $\mathbb{T} = \mathbb{T}^T$ symmetrical. Sometimes $P$ and consequently $\mathbb{T}$ too depend upon $\mathbf{u}$, contrary to our assumptions. None of these possibilities affect the trick.

Finally, the $j^{th}$ row $(\mathbb{T}\cdot\mathbf{u})_j$ of $\mathbb{T}\cdot\mathbf{u}$ always has the form $a_{j-1}\cdot\mathbf{u}_{j-1} - b_j\cdot\mathbf{u}_j + c_{j+1}\cdot u_{j+1}$ in which the coefficients $a_{j-1}$, $b_j$ and $c_{j+1}$ have these three properties:

**0)** $b_j = a_{j-1} + c_{j+1}$ for every $j$ except possibly $j = 0$ and/or $j = N$.

**1)** Both $|\mathbf{q}_j|/(|a_{j-1}| + |b_j| + |c_{j+1}|) \to 0$ and $|\mathbf{r}_j|/(|a_{j-1}| + |b_j| + |c_{j+1}|) \to 0$ roughly like $(h_{j-1} + h_j)\cdot\max\{h_{j-1}, h_j\} \to 0$.

**2)** $a_{j-1}/c_{j+1} = (p_{j-1/2}/h_{j-1})/(p_{j+1/2}/h_j)$ is near 1 for every $j$ except $j = 0$ and $j = N$.

Sins lurk in the words "roughly like" and "near". First, $N \to \infty$ when $\max\{h_{j-1}, h_j\} \to 0$, changing the meanings of the indices $j$. Second, gaps $h_j$ get shrunk in order to enhance the accuracy with which $\mathbf{u}_j$ approximates $U(x_j)$; but when shrinkage occurs adaptively some gaps shrink while others don't. Usually adjacent gaps differ by relatively little, as do adjacent values $p_{j\pm1/2}$ of the smooth function $P$; but occasional exceptions may violate property **2)**. None of these possibilities affect the trick.

**§5. How Roundoff Corrupts the Discretization:** It contributes uncertainty to almost every step of the solution process. Let $\varepsilon$ denote the roundoff threshold for rational floating-point arithmetic operations. When every operation rounds to the 24 sig. bits of `float`, $\varepsilon = 1/2^{24}$; … to 53 sig. bit `double`, $\varepsilon = 1/2^{53}$. Then rounded values computed from expressions like "v·w", "v/w", "v+w" and "v–w" lie in the respective ranges $(v\cdot w)\cdot(1\pm\varepsilon)$, $(v/w)\cdot(1\pm\varepsilon)$, $(v+w)/(1\pm\varepsilon)$ and $(v-w)/(1\pm\varepsilon)$. The smaller is $\varepsilon$, the higher is the arithmetic's precision, and then the smaller is the perturbation by roundoff of a finally computed result and thus the higher is its accuracy.

The discretization's first rounding errors corrupt at least the diagonal of $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$, and then more of them turn its factorization into $E\cdot B = \mathbb{T} + \mathrm{Diag}(\mathbf{q}) \pm \varepsilon\cdot\big(|E|\cdot|B| + |\mathbb{T}| + \mathrm{Diag}(|\mathbf{q}|)\big)$ roughly; the uncertainty here is dominated by the contributions from $\pm\varepsilon\cdot(|E|\cdot|B| + |\mathbb{T}|)$, the more so as the gaps $h_j$ shrink. If they shrink too far they can become so small that this uncertainty becomes comparable with or bigger than the separation between $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ and its nearest singular (*i.e.* non-invertible) matrix, thus rendering the solution process unpredictable. Though this extremity is rarely approached in practice it will be illustrated by the example in §10. Additional rounding

errors incurred during the forward and backward substitutions that solve $E\cdot B\cdot\mathbf{u} = \mathbf{r}$ for $\mathbf{u}$ will be ignored since they merely augment somewhat the uncertainties taken into account already. The bottom line: The first $\mathbf{u}$ computed satisfies $\big(\mathbb{T} + \mathrm{Diag}(\mathbf{q}) \pm \varepsilon\cdot\big(|E|\cdot|B| + |\mathbb{T}| + \mathrm{Diag}(|\mathbf{q}|)\big)\big)\cdot\mathbf{u} = \mathbf{r}$, not $(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \mathbf{r}$ as desired. We can ignore the difference if $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ is far enough from singular and if $\varepsilon$ is small enough, as is usually the case when the data's and arithmetic's precision extravagantly exceeds the accuracy desired in $\mathbf{u}$. I wish this were always the case. Life is so much simpler when roundoff can be ignored.

If roundoff's effect upon the first $\mathbf{u}$ computed cannot be ignored its accuracy must be improved by iterative refinement: Compute residual $\mathbf{s} := \mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$ and then solve $E\cdot B\cdot\Delta\mathbf{u} = \mathbf{s}$ for the correction that updates $\mathbf{u}$ to $\mathbf{u}+\Delta\mathbf{u}$. Other reasons for iterative refinement have been listed above. Other iterative methods solve for $\mathbf{u}$ without ever computing factors $E$ and $B$.

Every iterative method computes successive residuals $\mathbf{s} := \mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$ for updated values of $\mathbf{u}$ whose ultimate accuracy turns out to be limited mainly by how accurately $\mathbf{s}$ is computed. Its accuracy will be appraised next:

To match the magnitudes of the elements $\mathbf{r}_j$ and $\mathrm{Diag}(\mathbf{q}_j)$ in the equation $\mathbb{T}\cdot\mathbf{u} + \mathrm{Diag}(\mathbf{q})\cdot\mathbf{u} = \mathbf{r}$ that has to be solved for $\mathbf{u}$, the $j^{\text{th}}$ element $(\mathbb{T}\cdot\mathbf{u})_j := a_{j-1}\cdot\mathbf{u}_{j-1} - b_j\cdot\mathbf{u}_j + c_{j+1}\cdot\mathbf{u}_{j+1}$ has to suffer massive subtractive cancellation, the more so as gaps $h_{j-1}$ and $h_j$ shrink to help approximate $U(x_j)$ better by $\mathbf{u}_j$. Normally, when all the gaps are small, the computed value of this $(\mathbb{T}\cdot\mathbf{u})_j$ must cancel down to something small, roughly of the order of

$$(h_{j-1} + h_j)\cdot\max\{h_{j-1}, h_j\}\cdot\big(\,|a_{j-1}\cdot\mathbf{u}_{j-1}| + |b_j\cdot\mathbf{u}_j| + |c_{j+1}\cdot\mathbf{u}_{j+1}|\,\big) = (h_{j-1} + h_j)\cdot\max\{h_{j-1}, h_j\}\cdot(|\mathbb{T}|\cdot|\mathbf{u}|)_j$$

wherein the last two pairs of absolute value bars $|\ldots|$ are to be applied elementwise to $\mathbb{T}$ and $\mathbf{u}$.

Coincidentally, when $(\mathbb{T}\cdot\mathbf{u})_j$ is computed by evaluating "$a_{j-1}\cdot\mathbf{u}_{j-1} - b_j\cdot\mathbf{u}_j + c_{j+1}\cdot\mathbf{u}_{j+1}$" literally, roundoff contributes uncertainty of the order of $\pm\varepsilon\cdot(|\mathbb{T}|\cdot|\mathbf{u}|)_j$ to $(\mathbb{T}\cdot\mathbf{u})_j$. This term dominates the computed residual's uncertainty due to roundoff: $\mathbf{s} - \mathbf{r} + (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \pm\varepsilon\cdot(|\mathbb{T}| + \mathrm{Diag}(|\mathbf{q}|))\cdot|\mathbf{u}|$ roughly within a factor of 2 or 3 regardless of whether $(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$ is computed after $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ is, or computed separately as $\mathbb{T}\cdot\mathbf{u} + \mathrm{Diag}(\mathbf{q})\cdot\mathbf{u}$. Even if the computed residual vanishes the current estimate $\mathbf{u}$ must satisfy $(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \mathbf{r} \pm \varepsilon\cdot(|\mathbb{T}| + \mathrm{Diag}(|\mathbf{q}|))\cdot|\mathbf{u}|$, an equation perturbed by uncertainty dominated by the term $\pm\varepsilon\cdot|\mathbb{T}|\cdot|\mathbf{u}|$. This is almost as bad as the uncertainty $\pm\varepsilon\cdot\big(|E|\cdot|B| + |\mathbb{T}| + \mathrm{Diag}(|\mathbf{q}|)\big)\big)\cdot\mathbf{u}$ that afflicted Gaussian elimination and triangular factorization unless an unfortunate choice of pivots bloated the triangular factors $E$ and $B$. This dominant term $|\mathbb{T}|\cdot|\mathbf{u}|$ is bigger than all the other terms $\mathbf{r}$, $\mathrm{Diag}(|\mathbf{q}|)\cdot|\mathbf{u}|$ and even $|\mathbb{T}\cdot\mathbf{u}|$ by factors like $1/\big(\,(h_{j-1} + h_j)\cdot\max\{h_{j-1}, h_j\}\,\big)$ that increase when gaps get shrunk to reduce the difference between $\mathbf{u}$ and $U$ due to discretization. Then the equation $\mathbf{u}$ satisfies gets more perturbed.

The uncertainty in $\mathbf{u}$ due to roundoff is inconsequential when the precision of arithmetic and *all* intermediate variables exceeds extravagantly the precision (presumed the same) of the data $\mathbb{T}$, $\mathbf{q}$ and $\mathbf{r}$ and the accuracy desired from $\mathbf{u}$. A little more than twice as precise is usually extravagant enough. This is so because the uncertainty due to roundoff usually grows by a factor roughly the reciprocal of the factor by which the discretization error shrinks when all gaps $h_j$ are shrunk, and then the number of correct digits in the elements of $\mathbf{u}$ cannot much exceed half the digits carried

by the arithmetic. There are exceptions; cruder discretizations can lose a larger fraction of the digits carried; defter will lose a smaller fraction. Any such lost fraction of extravagantly too many digits will leave enough of them to produce adequately accurate results.

What is to be done when the available arithmetic's precision at most barely exceeds the data's precision and the desired accuracy? Our next objective is to trick the arithmetic into losing not some fraction like half the digits carried but at most a few of them so as to compute **u** about as accurately as the given data determine it. The trick is to compute residuals **s** well enough.

**§6. Accurate Residuals:** The trick is to compute residuals $\mathbf{s} := \mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$ obscured by small rounding errors of the order of $\varepsilon\cdot(|\mathbb{T}\cdot\mathbf{u}| + \mathrm{Diag}(|\mathbf{q}|)\cdot|\mathbf{u}| + |\mathbf{r}|)$ instead of huge rounding errors of the order of $\varepsilon\cdot(|\mathbb{T}|\cdot|\mathbf{u}| + \mathrm{Diag}(|\mathbf{q}|)\cdot|\mathbf{u}| + |\mathbf{r}|)$. It's about small $|\mathbb{T}\cdot\mathbf{u}|$ versus huge $|\mathbb{T}|\cdot|\mathbf{u}|$.

For definiteness let us assume all the given data, namely $\{x_j\}$ and columns **q** and **r** and arrays $\{a_j\}$ and $\{c_j\}$ of off-diagonal elements of $\mathbb{T}$, and the columns **u** of a putative solution and its residual **s**, to be stored in the computer's memory as seven arrays of 4-byte-wide `floats`. If we use triangular factors to solve the equations for **u** and $\Delta\mathbf{u}$ then we store also the two arrays of subdiagonal elements of E and diagonal elements of B as `floats` and write `float` $\Delta\mathbf{u}$ over **s**.

The array $\mathbf{g} := \mathrm{diag}(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))$ can be treated in any of several ways: One is to compute each element $\mathbf{g}_j := \mathbf{q}_j - a_{j-1} - c_{j+1}$ at the moment of need using array elements $a_{j-1}$ and $c_{j+1}$ that will be needed at the same moment. Another way is to compute in advance and and store the array **g** as an array of 8-byte `doubles`. Trickier ways to cope with **g** will be passed over for the sake of a simpler exposition.

In a benign computing environment, as was provided by the original Kernighan-Ritchie ***C*** and is now available from some implementations of ***C***99, every arithmetic operation is rounded to at least `double` regardless of whether its operands are `floats` or `doubles`. In this environment the `float` residual **s** can be computed amply accurately from any one of the three assignments

" $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot\mathbf{u}_{j-1} - \mathbf{g}_j\cdot\mathbf{u}_j - c_{j+1}\cdot\mathbf{u}_{j+1}$ " or

" $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot\mathbf{u}_{j-1} - (\mathbf{q}_j - a_{j-1} - c_{j+1})\cdot\mathbf{u}_j - c_{j+1}\cdot\mathbf{u}_{j+1}$ " or

" $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot(\mathbf{u}_{j-1} - \mathbf{u}_j) - c_{j+1}\cdot(\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j\cdot\mathbf{u}_j$ "

each of whose right-hand side's every arithmetic operation is rounded to `double` before being stored as a `float` in $\mathbf{s}_j$. Thus is adequate accuracy achieved with no extra effort nor thought.

In a FORTRANnish environment like JAVA or ANSII ***C*** (1987), the foregoing assignments must be encumbered by *casts* (conversions to `double`) to achieve amply adequate accuracy thus:

" $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot(\texttt{double})\mathbf{u}_{j-1} - \mathbf{g}_j\cdot(\texttt{double})\mathbf{u}_j - c_{j+1}\cdot(\texttt{double})\mathbf{u}_{j+1}$ " or

" $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot(\texttt{double})\mathbf{u}_{j-1} - (((\texttt{double})\mathbf{q}_j - a_{j-1}) - a_j)\cdot\mathbf{u}_j - c_{j+1}\cdot(\texttt{double})\mathbf{u}_{j+1}$ " or

" $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot((\texttt{double})\mathbf{u}_{j-1} - \mathbf{u}_j) - c_{j+1}\cdot((\texttt{double})\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j\cdot(\texttt{double})\mathbf{u}_j$ ".

(HONOR PARENTHESES!)

**§7. A Trickier Trick:** In a benighted environment where `double` is too slow or inconvenient, or unavailable, barely adequate accuracy may be achieved at the cost of two extra subtractions:

$$\text{“ } \mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot((\mathbf{u}_{j+1} - \mathbf{u}_j) - (\mathbf{u}_j - \mathbf{u}_{j-1})) - (c_{j+1} - a_{j-1})\cdot(\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j\cdot\mathbf{u}_j \text{ ”} .$$

Why does this tricky trick work when it works?

It doesn't work unless $c_{j+1}$ and $a_{j-1}$ are close enough, and this requires typically that the gaps $h_j$ and $h_{j-1}$ be equal or almost equal according to property 2) above. In such cases we expect all but a few of the quotients $a_{j-1}/c_{j+1}$ and $\mathbf{u}_{j-1}/\mathbf{u}_j$ to stay close to 1, and to come closer as all the gaps $h_j$ shrink. Then each subtraction $c_{j+1} - a_{j-1}$, $\mathbf{u}_{j+1} - \mathbf{u}_j$, and usually $(\mathbf{u}_{j+1} - \mathbf{u}_j) - (\mathbf{u}_j - \mathbf{u}_{j-1})$ incurs substantial cancellation *but no new rounding error.* For a more quantitative appraisal of the extra subtractions' attenuation of roundoff see Appendix 1 below.

If the grid's gaps $h_j$ vary more than minimally, a different and far trickier trick will be needed to compute the residual $\mathbf{s}$ accurately enough. This trickier trick requires that all gaps $h_j$ be powers of $1/2$ to ensure that multiplications and divisions by gaps incur no new rounding errors. This requirement is less onerous than first appears: It requires first that the independent variable $x$ be scaled (multiplied or divided) to turn the domain of $U(x)$ into an interval whose width $x_N - x_0$ is an integer multiple of a power of $1/2$. Secondly, after the initial distribution of grid points ensures that every gap $h_j := x_{j+1} - x_j$ is a power of $1/2$, subsequent grid refinements will plant new mesh-points only halfway between adjacent previously planted mesh-points.

Now we shall appraise roundoff's intrusion into the discretization of $(P\cdot U')'$ in §4, namely

$$2\cdot( p_{j+1/2}\cdot(\mathbf{u}_{j+1} - \mathbf{u}_j)/h_j - p_{j-1/2}\cdot(\mathbf{u}_j - \mathbf{u}_{j-1})/h_{j-1} )/(h_j + h_{j-1}) \approx (P\cdot U')' .$$

To simplify its appraisal we suppose that $P(x)$, $U(x)$ and their first two derivatives' magnitudes are all of order 1, huge compared with the roundoff threshold $\varepsilon$ and the gaps $h_j$ all of order $h$, say. Then $(\mathbf{u}_{j+1} - \mathbf{u}_j)/h_j$ is of order 1 because it approximates $U'(x_j)$ roughly; the quotient introduces no *new* roundoff. The multiplication in $p_{j+1/2}\cdot(\mathbf{u}_{j+1} - \mathbf{u}_j)/h_j$ suffers a new rounding error of order $\varepsilon$, as does the other multiplication, so the foregoing discretization of $(P\cdot U')'$ differs from it by the formula's discretization error, at most $O(h)$, plus a contribution of order $\varepsilon/h$ from roundoff.

Reducing the last contribution from order $\varepsilon/h$ to order $\varepsilon$ is the trickier trick's goal. It changes the discretization's formula to an algebraically equivalent but more complicated new formula

$$2\cdot\big( p_{j+1/2}\cdot( (\mathbf{u}_{j+1} - \mathbf{u}_j)/h_j - (\mathbf{u}_j - \mathbf{u}_{j-1})/h_{j-1} ) + ( p_{j+1/2} - p_{j-1/2} )\cdot(\mathbf{u}_j - \mathbf{u}_{j-1})/h_{j-1} \big)/(h_j + h_{j-1})$$

that costs two extra subtractions neither of which introduces a new rounding error. The first extra subtraction $(\mathbf{u}_{j+1} - \mathbf{u}_j)/h_j - (\mathbf{u}_j - \mathbf{u}_{j-1})/h_{j-1} \approx U'(x_j + h_j/2) - U'(x_j - h_{j-1}/2) \approx U''(xj)\cdot(h_j + h_{j-1})/2$ very roughly, so it is of order $h$ and the subsequent multiplication suffers a rounding error of order $h\cdot\varepsilon$. Similarly the second extra subtraction $p_{j+1/2} - p_{j-1/2} \approx P'(x_j)\cdot(h_j + h_{j-1})/2$ and its subsequent multiplication inject another rounding error of order $\varepsilon\cdot h$. The subsequent addition and then division by $(h_j + h_{j-1})/2$ make the new formula's total contribution from roundoff of order $\varepsilon$, not $\varepsilon/h$.

Thus are computed residuals $\mathbf{s} := \mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$ contaminated by tiny practically irreducible rounding errors of the order of $\varepsilon\cdot(|\mathbb{T}|\cdot|\mathbf{u}| + \mathrm{Diag}(|\mathbf{q}|)\cdot|\mathbf{u}| + |\mathbf{r}|) \approx O(\varepsilon)$ instead of huge rounding errors of the order of $\varepsilon\cdot(|\mathbb{T}|\cdot|\mathbf{u}| + \mathrm{Diag}(|\mathbf{q}|)\cdot|\mathbf{u}| + |\mathbf{r}|) \approx O(\varepsilon/h^2)$ as all gaps shrink towards zero. This attenuation of roundoff's contamination of the residuals $\mathbf{s}$ is rewarded by a consequent attenuation of roundoff's contamination of the computed solution $\mathbf{u}$ of the discretized problem.

Similar tricks enhance the accuracy of $\mathbf{u}$ when divided-difference formulas of higher order in the gaps $h_{\cdots}$ are used to approximate the differential equation; see Appendix 2 below. Similar tricks enhance the accuracy of computed solutions of elliptic partial differential equations like $\mathrm{div}(\boldsymbol{p}\bullet\mathbf{grad}\, u) + q\cdot u = r$. Similar tricks are applicable to some *Finite-Element* discretizations.

Tricky tricks admit innumerable opportunistic variations. We will not attempt to patent them all. More important is the realization that they would be rendered unnecessary by a simple expedient:

> Routinely (by default) perform *all* arithmetic and carry *all* intermediate variables extravagantly more precisely than the data and the accuracy desired in computed results.

**§8. Example:** The singular differential equation $(x\cdot u')' + 4x\cdot(1-x^2)\cdot u = 0$ has regular solutions all with $u'(0) = 0$ and so $u(-x) \equiv u(x)$. We wish to compute the regular solution satisfying the boundary conditions $u(\pm1) = 1$ as if we did not know that $u(x) = \exp(1-x^2)$. The numerical estimation of $u(x)$ is complicated by the differential equation's singular solutions

$$v(x) := C\cdot\exp(-x^2)\cdot\int \exp(2x^2)\cdot dx/x \;=\; C\cdot\exp(-x^2)\cdot\Big(\ln(|x|) - \int_{|x|}^{1} (\exp(2\xi^2) - 1)\cdot d\xi/\xi\Big)\,.$$

Their constants $C$ can be different for $x > 0$ than for $x < 0$. All have a logarithmic pole at $x = 0$. The pole can amplify tiny perturbations of the differential equation into a narrow spike at $x = 0$. Worse, this singular solution $v$ satisfies $v(-x) \equiv v(x)$ and $v(\pm1) = 0$ and the differential equation except at $x = 0$, so a discretized analog of this $v(x)$ can contaminate the numerical approximation of the regular solution $u(x)$ unless filtered out.

Filtering won't affect the trick. Then why choose to illustrate it applied to a singular differential equation instead of something simpler? Because of spikes. They are often misdiagnosed, blamed upon the differential equation's singularity instead of roundoff, or *vice-versa*. Spikes of both kinds will afflict this example, and our analysis will distinguish them and then eliminate them.

Choose a large integer $N >> 2$ and set $x_j := j/N - 1$ for $j := 0, 1, 2, \ldots, 2N-1, 2N$. Now every $h_j := 1/N$, $q_j := 4x_j\cdot(1-x_j)\cdot(1+x_j) = 4j\cdot(N-j)\cdot(j-2N)/N^3 = -q_{2N-j}$ and $p_{j+1/2} := (j+1/2)/N - 1$. The numerical estimates $\mathbf{u}_j$ of $u(x_j)$ satisfy discretized equations $a_{j-1}\cdot\mathbf{u}_{j-1} + g_j\cdot\mathbf{u}_j + c_{j+1}\cdot\mathbf{u}_{j+1} = 0$ in which $c_j := a_{j-1} := N^2\cdot p_{j-1/2} = -N\cdot(N-j+1/2) = -a_{2N-j}$ and $g_j := q_j - a_{j-1} - c_{j+1} = -g_{2N-j}$. The discretization error is $a_{j-1}\cdot u(x_{j-1}) + g_j\cdot u(x_j) + c_{j+1}\cdot u(x_{j+1}) \approx (2u'''(x_j) + x_j\cdot u''''(x_j))/(12N^2)$. We seek solutions $\mathbf{u}_j$ of these discretized equations satisfying boundary conditions $\mathbf{u}_0 = \mathbf{u}_{2N} = 1$.

A first attempt constructs a symmetric tridiagonal matrix $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ with $[g_1, g_2, \ldots, g_{2N-1}]$ on its main diagonal and $[c_2, c_3, \ldots, c_{2N-1}] = [a_1, a_2, \ldots, a_{2N-2}]$ on its first superdiagonal and first

subdiagonal. Every element of column $\mathbf{r}$ is zero except for its first element $r_1 := -a_0 \cdot \mathbf{u}_0 = -a_0$ and its last $\mathbf{r}_{2N-1} = -c_{2N} \cdot \mathbf{u}_{2N} = -a_{2N-1} = a_0$ that convey the boundary conditions $u(\pm 1) := 1$.

Elements $[\mathbf{u}_1; \mathbf{u}_2; \ldots; \mathbf{u}_{2N-1}]$ of column $\mathbf{u}$ should be computed by solving $(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$. The first attempt factorizes $\mathbb{T} + \mathrm{Diag}(\mathbf{q}) = \mathrm{E} \cdot \mathrm{B}$ wherein B is bidiagonal with $[ß_1, ß_2, \ldots, ß_{2N-1}]$ on its diagonal and $[a_1, a_2, \ldots, a_{2N-2}]$ on its first superdiagonal, and E is bidiagonal with 1 everywhere on its diagonal and $[e_1, e_2, \ldots, e_{2N-2}]$ on its first subdiagonal. A recurrence produces $e_{j-1} := a_{j-1}/ß_{j-1}$ and $ß_j := g_j - a_{j-1} \cdot e_{j-1}$ for $j = 2, 3, \ldots, 2N-1$ in turn starting with $ß_1 := g_1$. This amounts to Gaussian elimination without pivotal exchanges. Its simplicity of programming could be offset by the appearance of a tiny $ß_{j-1}$ followed by a huge $ß_j$ that greatly amplifies roundoff. Numerical degradation like that has yet to occur during the foregoing computation. However the last $ß_{2N-1} = 0$ except for roundoff because $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ is singular (not invertible), which complicates the first attempt to compute $\mathbf{u}$.

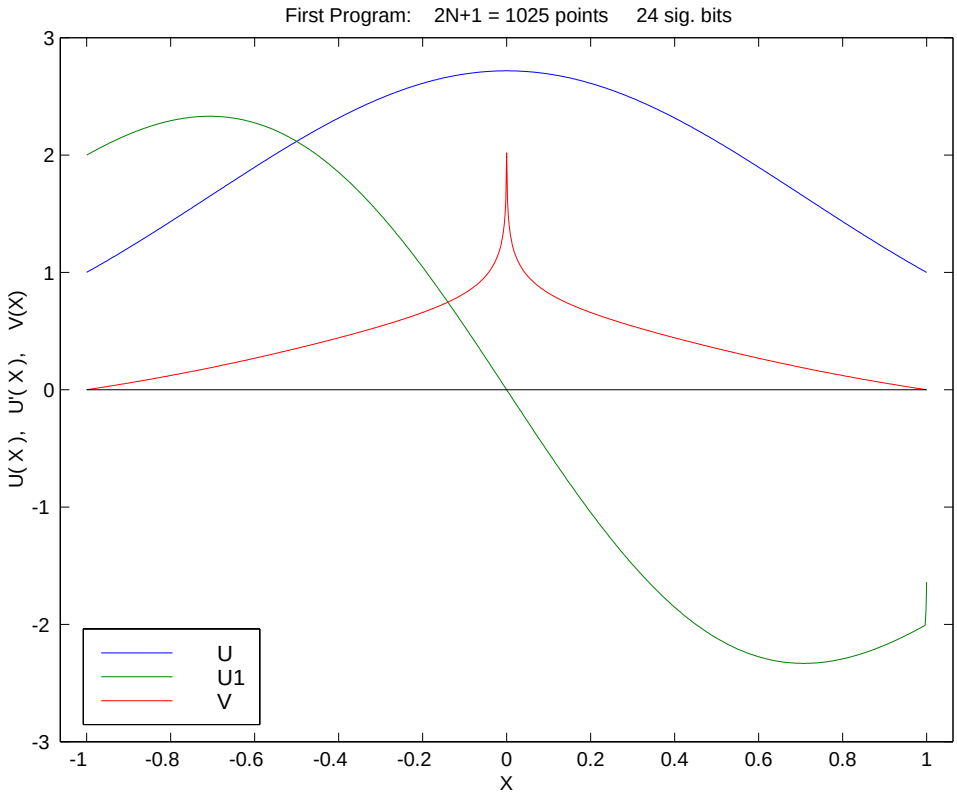
To see why $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ is singular let $\Xi = \Xi^{\mathrm{T}}$ be the matrix obtained from the 2N–1 -by- 2N–1 identity by reversing the order of its rows (or columns) and observe that $\Xi \cdot (\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \Xi = -(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))$, whereupon $\det(\Xi)^2 \cdot \det(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) = (-1)^{2N-1} \cdot \det(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) = 0$. Then $\det(\mathrm{B}) = \det(\mathrm{E}) \cdot \det(\mathrm{B}) = \det(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) = 0$ too, which explains why $ß_{2N-1} = 0$. Despite that $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ is singular, the equation $(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ turns out to be consistent. All its solutions are symmetric, $\mathbf{u} = \Xi \cdot \mathbf{u}$, though all of them but one are contaminated by some scalar multiples of a singular solution $\mathbf{v} = \Xi \cdot \mathbf{v} \neq \mathbf{o}$ of $(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \mathbf{v} = \mathbf{o}$ with a sharp spike in the middle of it.

Any first attempt to solve $(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ naively gets $\mathbf{u}$ contaminated by the addition of some arbitrary or infinite multiple of the spiked singular solution $\mathbf{v}$ of $(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \mathbf{v} = \mathbf{o} \neq \mathbf{v}$. If that multiple is small enough, the spike is narrow enough to go unnoticed until too late. How can the smooth regular solution $u(\mathrm{x})$ be separated numerically from spiked singular solutions $v(\mathrm{x})$ that satisfy the same boundary conditions and even the symmetry condition $v(-\mathrm{x}) \equiv v(\mathrm{x})$ ?

There is one way: A regular solution's $u'(0) = 0$ differs from a singular solution's $v'(0) = \pm\infty$ utterly, whence *l'Hôpital's Rule* implies $u'(\mathrm{x})/\mathrm{x} \to u''(0)$ as $\mathrm{x} \to 0$; and then substitution into the differential equation implies $u''(0) = -2u(0)$. This *internal boundary condition* upon $u(\mathrm{x})$ further distinguishes it from all singular solutions. Discretized, this internal boundary condition turns into $N^2 \cdot (\mathbf{u}_{N+1} - 2\mathbf{u}_N + \mathbf{u}_{N-1}) = -2\mathbf{u}_N$ which, if not satisfied by a computed solution $\hat{\mathbf{u}}$ of $(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \hat{\mathbf{u}} = \mathbf{r}$, can be imposed upon a revised solution $\mathbf{u} := \hat{\mathbf{u}} - \lambda \cdot \mathbf{v}$ by choosing $\lambda$ aptly after computing a singular solution $\mathbf{v} \neq \mathbf{o}$ of $(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) \cdot \mathbf{v} = \mathbf{o}$. Our first program does this.

**1st Program's Details:** The computed solution of $\mathrm{B} \cdot \hat{\mathbf{u}} = \mathrm{E}^{-1} \cdot \mathbf{r}$ is $\hat{\mathbf{u}}$; its last component is set to $\hat{\mathbf{u}}_{2N-1} := 1 + 2/N$ to filter out most of the singular solution $\mathbf{v}$. This is the solution of $\mathrm{B} \cdot \mathbf{v} = \mathbf{o}$; its last component $\mathbf{v}_{2N-1} := 1/(2N-1)$ is set to keep the spike $\mathbf{v}_N$ roughly between 1 and 4. Then $\lambda$ comes close to $-2/N$ when computed to make $\mathbf{u} := \hat{\mathbf{u}} - \lambda \cdot \mathbf{v}$ satisfy the discretized internal boundary condition. Apparently $\mathbf{u}$ loses little to cancellation in the last subtraction. Then the gradient $u'(\mathrm{x}_j)$ is approximated within $O(1/N^2)$ by $\mathbf{u}^{\ddagger}_j := (\mathbf{u}_{j+1} - \mathbf{u}_{j-1}) \cdot N/2$ except at the boundarys where $u'(-1) \approx \mathbf{u}^{\ddagger}_0 := (4\mathbf{u}_1 - 3\mathbf{u}_0 - \mathbf{u}_2) \cdot N/2$. The errors $\max_j |\mathbf{u}_j - u(\mathrm{x}_j)|$ in $\mathbf{u}$ and $\max_j |\mathbf{u}^{\ddagger}_j - u'(\mathrm{x}_j)|$ in $\mathbf{u}^{\ddagger}$ would roughly approximate $2.4/N^2$ and $3.7/N^2$ respectively for large N if they were due to discretization alone, but something else happens when roundoff contaminates the whole process.

Computed Graphs of $\mathbf{u} \approx u(x)$, $\mathbf{u}^{\ddagger} \approx u'(x)$ and $\mathbf{v} \approx v(x)$ carrying 24 sig. bits




Results from the First Program carrying 24 sig. bits ( $\varepsilon \approx 6/10^8$ )

| N | err(**u**) | err(**u**)·N$^2$ | err(**u**$^{\ddagger}$) | err(**u**$^{\ddagger}$)·N$^2$ |
|---|---|---|---|---|
| 16 | 0.009324 | 2.39 | 0.01529 | 3.9 |
| 24 | 0.004144 | 2.39 | 0.00666 | 3.8 |
| 32 | 0.002326 | 2.38 | 0.00366 | 3.7 |
| 48 | 0.001040 | 2.40 | 0.00162 | 3.7 |
| 64 | 0.000568 | 2.33 | 0.00089 | 3.6 |
| 96 | 0.000398 | 3.67 | 0.00461 | 42.5 |
| 128 | 0.000128 | 2.10 | 0.00230 | 37.6 |
| 192 | 0.000527 | 19.43 | 0.01331 | 490.8 |
| 256 | 0.000093 | 6.11 | 0.03575 | 2343.0 |
| 384 | 0.000453 | 66.87 | 0.03584 | 5284.1 |
| 512 | 0.000472 | 123.72 | 0.36157 | 94784.0 |
| 768 | 0.002734 | 1612.64 | 0.69930 | 412461.0 |

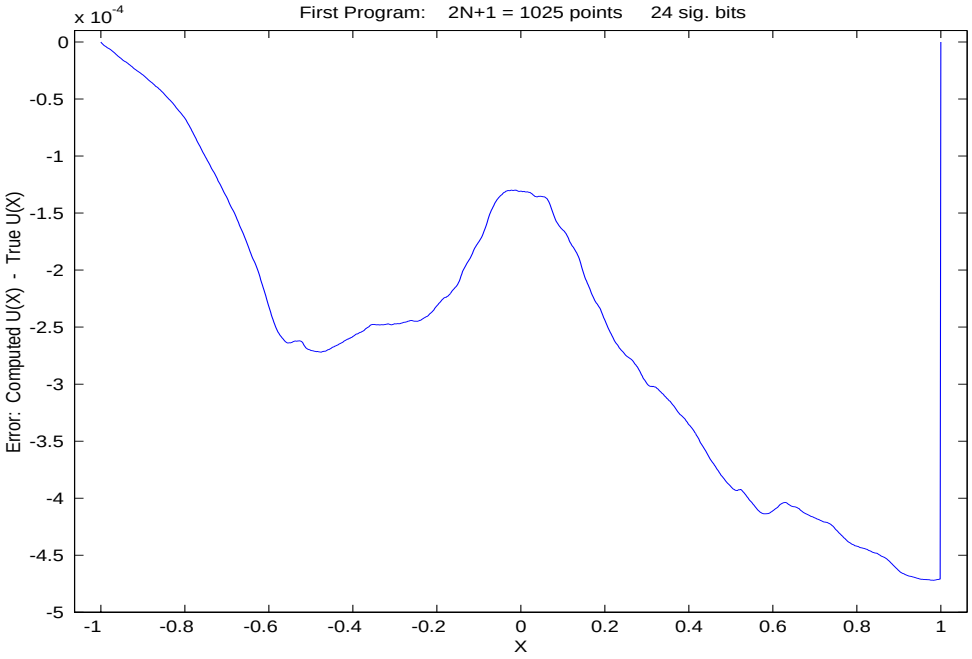
N = #gaps, err(**u**) := $\max_j |\mathbf{u}_j - u(x_j)|$ , err(**u**$^{\ddagger}$) := $\max_j |\mathbf{u}^{\ddagger}_j - u'(x_j)|$ .

The foregoing program loses accuracy to roundoff about as badly as the preceding analysis had predicted, almost as badly as if the roundoff threshold $\varepsilon$ had been magnified to $\varepsilon \cdot N^2$. The error in **u** due to discretization alone, roughly $2.4/N^2$, decreases as N increases but the total error in

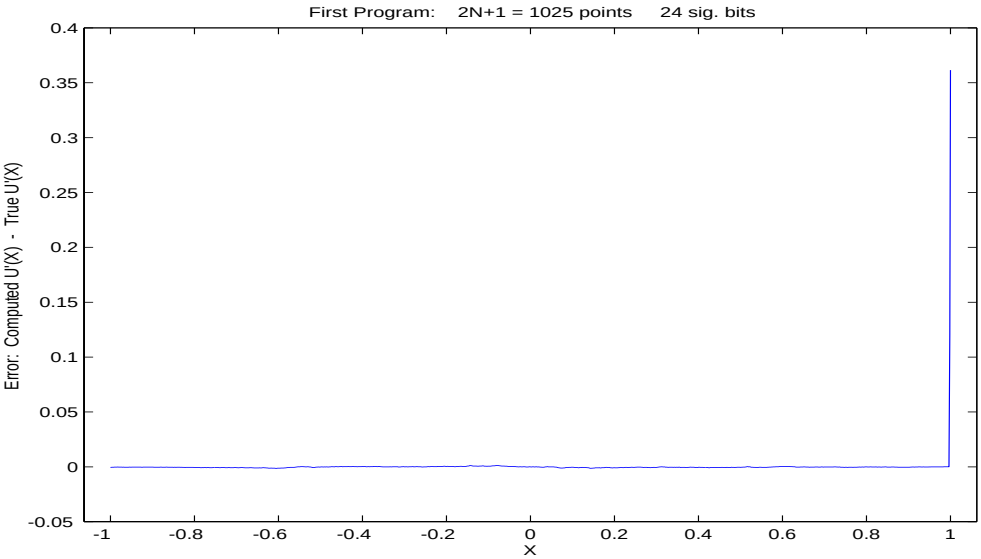
$\mathbf{u}$ never gets much below about $\sqrt{\varepsilon}$, and making N too big actually worsens the total error by amplifying roundoff. Usually. Roundoff contributes raggedly (*not randomly*) to the total error.

The gradient's estimate $\mathbf{u}^{\ddagger}$ is damaged worst by the destruction of the symmetry $u(-x) \equiv u(x)$ by roundoff, which tends to pile up towards the end at $\mathbf{u}_{2N-1}$, generating a huge spike in the error $\mathbf{u}^{\ddagger}_{2N} - u'(1)$ overwhelmingly bigger than every other error $\mathbf{u}^{\ddagger}_{j} - u'(x_j)$ for $0 \le j < 2N$.

Error in the First Program's $\mathbf{u} \approx u(x)$ carrying 24 sig. bits, N = 512




Error in the First Program's $\mathbf{u}^{\ddagger} \approx u'(x)$ carrying 24 sig. bits, N = 512




This spike at x = 1 is *not* caused by the differential equation's singularity at x = 0. Instead the spike is caused by the hypersensitivity to roundoff of our first program's numerical method, as is confirmed when the same program is rerun carrying everywhere 53 sig. bits instead of 24.

By far the simplest remedy for the program's loss of about half the sig. digits carried is to declare all variables to be 8-byte `doubles` and round all arithmetic to `double`'s 53 sig. bits or more. Then the error in $\mathbf{u}$ can be reduced below $10^{-9}$ and the error in $\mathbf{u}^{\ddagger}$ below $10^{-6}$ if N exceeds roughly $2^{15}$. Increasing N too far beyond this loses to roundoff at least about half the 53 sig. bits carried by `double`'s arithmetic.

To achieve accuracy more nearly commensurate with the precision of all variables and arithmetic, the program must incorporate iterative refinement using residuals computed accurately enough. "Accurately enough" requires the trick. To expose its benefits fairly, the program must first be simplified by the use of symmetry to eradicate singular solutions $\mathbf{v}$ and the spike in $\mathbf{u}^{\ddagger}$'s error.

**§9. The 2nd Program:** After the symmetry condition $\mathbf{u}_{N+j} = \mathbf{u}_{N-j}$ is applied, the internal boundary condition derived above becomes $N^2\cdot\mathbf{u}_{N-1} - (N^2-1)\cdot\mathbf{u}_N = 0$. That symmetry halves the work needed to compute the desired solution $u(x) \equiv u(-x)$; it need be computed only for $-1 < x \le 0$, whence $\mathbf{u}_j \approx u(x_j)$ need be computed only for $1 \le j \le N$.

How do we know that *every* regular solution $u$ possesses the symmetry $u(-x) \equiv u(x)$ ? There are two ways to prove it: One way computes the power series expansion of $u(x) = \sum_{n\ge 0} \mu_n\cdot x^n/n!$ starting with an arbitrary $\mu_0 = u(0) \ne 0$ though $\mu_1 = u'(0) = 0$; the other coefficients $\mu_n$ are obtained by recurrence after the series is substituted into the differential equation. Doing so establishes that every $\mu_{2n+1} = 0$, whence follows $u(-x) \equiv u(x)$. A less laborious but more devious proof starts from the observation that, if $u(x)$ is a regular solution of $(x\cdot u')' + 4x\cdot(1-x^2)\cdot u = 0$ then so are $u(-x)$ and $w(x) := u(x) - u(-x) \equiv -w(-x)$; moreover $w(0) = w'(0) = 0$. Suppose, for the sake of an argument by contradiction, that $w(x) \ne 0$ somewhere. Where might $w'(x)$ vanish in the open interval $0 < x < 1$ ? If nowhere then set $\zeta := 1$; otherwise let $x = \zeta$ be the least zero of $w'(x)$ in that interval. Then $w(x) = \int_0^x w'(\xi)d\xi$ would have the same sign, say positive, as $w'(x)$ has for $0 < x < \zeta \le 1$. And then we would find that

$$0 < \int_0^\zeta 4x\cdot(1-x^2)\cdot w(x)dx = -\int_0^\zeta (x\cdot w'(x))'dx = -\zeta\cdot w'(\zeta) \le 0 .$$ This is impossible.

Consequently $w(x) = 0$ at least for $0 \le x \le 1$; beyond $x = 1$ the differential equation is regular and determines its solution $w(x) = 0$ uniquely for all $x \ge 0$ from $w(1) = w'(1) = 0$. Therefore regular solutions $u$ are symmetrical; $u(x) \equiv u(-x)$ is determined uniquely by $u'(0) = 0$ at its internal boundary and by $u(\pm1) = 1$ at an external boundary.

This uniqueness has important numerical consequences. It implies that the system of linear equations, obtained by adjoining the internal boundary condition's discretization to the differential equation's, defines its solution uniquely; so its new matrix $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ must be invertible and far enough from singular that the consequences of roundoff will become negligible if it is kept small enough.

Here is the scheme simplified by symmetry: Choose a big integer $N > 2$ and set $x_j := j/N - 1$ for $j := 0, 1, 2, \ldots, N-1, N$. Again every $h_j := 1/N$, $\mathbf{q}_j := 4x_j\cdot(1-x_j)\cdot(1+x_j) = 4j\cdot(N-j)\cdot(j-2N)/N^3$ and $p_{j+1/2} := (j+1/2)/N - 1$. The numerical estimates $\mathbf{u}_j$ of $u(x_j)$ satisfy discretized equations $a_{j-1}\cdot\mathbf{u}_{j-1} + g_j\cdot\mathbf{u}_j + c_{j+1}\cdot\mathbf{u}_{j+1} = 0$ in which $c_j := a_{j-1} := N^2\cdot p_{j-1/2} = -N\cdot(N-j+1/2)$ again but now for $1 \le j \le N$, and $g_j := \mathbf{q}_j - a_{j-1} - c_{j+1}$ again but now for $1 \le j \le N-1$. The discretized internal boundary condition $N^2\cdot\mathbf{u}_{N-1} = (N^2-1)\cdot\mathbf{u}_N$ is effected by setting $g_N := N/2 - 1/(2N)$ and then discarding $c_{N+1}\cdot u_{N+1}$. The external boundary condition is $\mathbf{u}_0 := 1$.

Our second attempt to compute the elements $[\mathbf{u}_1; \mathbf{u}_2; \ldots; \mathbf{u}_N]$ of column $\mathbf{u}$ constructs a new symmetric tridiagonal matrix $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ with $[g_1, g_2, \ldots, g_{N-1}, g_N]$ on its main diagonal and $[c_2, c_3, \ldots, c_N] = [a_1, a_2, \ldots, a_{N-1}]$ on its first superdiagonal and first subdiagonal. Every element of column $\mathbf{r}$ is zero except for its first element $r_1 := -a_0 \cdot \mathbf{u}_0 = -a_0$ that conveys the boundary condition $u(\pm 1) := 1$ .

To solve $(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \mathbf{r}$ we first factorize $\mathbb{T} + \mathrm{Diag}(\mathbf{q}) = \mathrm{E}\cdot\mathrm{B}$ wherein B is bidiagonal with $[ß_1, ß_2, \ldots, ß_N]$ on its diagonal and $[a_1, a_2, \ldots, a_{N-1}]$ on its first superdiagonal, and E is bidiagonal with 1 everywhere on its diagonal and $[e_1, e_2, \ldots, e_{N-1}]$ with $e_j := a_j/ß_j$ on its first subdiagonal. The numbers $ß_j$ are computed from a recurrence $ß_j := g_j - a_{j-1}\cdot e_{j-1}$ for $j = 2, 3, \ldots, N$ in turn starting with $ß_1 := g_1$ . This amounts to the same Gaussian elimination without pivotal exchanges as before; however, since the new $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ turns out to be positive definite there is no risk now that some tiny pivot $ß_{j-1}$ will be followed by an enormous $ß_j$ .

Forward substitution computes $\mathbf{w} := \mathrm{E}^{-1}\cdot\mathbf{r}$ : $\mathbf{w}_1 := r_1$ and $\mathbf{w}_j := -e_{j-1}\cdot\mathbf{w}_{j-1}$ . Subsequent back-substitution computes $\mathbf{u} := \mathrm{B}^{-1}\cdot\mathbf{w}$ : $\mathbf{u}_N := \mathbf{w}_N/ß_N$ and $\mathbf{u}_j := (\mathbf{w}_j - a_j\cdot\mathbf{u}_{j+1})/ß_j$ . Finally, gradient $u'(x)$ is approximated by $\mathbf{u}^{‡}$ as before.

Results from the Second Program carrying 24 sig. bits ( $\varepsilon \approx 6/10^8$ )

| N | err($\mathbf{u}$) | err($\mathbf{u}$)·$N^2$ | err($\mathbf{u}^{‡}$) | err($\mathbf{u}^{‡}$)·$N^2$ |
|---|---|---|---|---|
| 16 | 0.009324 | 2.39 | 0.01530 | 3.9 |
| 24 | 0.004146 | 2.39 | 0.00662 | 3.8 |
| 32 | 0.002326 | 2.38 | 0.00365 | 3.7 |
| 48 | 0.001028 | 2.37 | 0.00158 | 3.6 |
| 64 | 0.000663 | 2.73 | 0.00099 | 4.0 |
| 96 | 0.000118 | 1.09 | 0.00022 | 2.0 |
| 128 | 0.000073 | 1.19 | 0.00027 | 4.5 |
| 192 | 0.000531 | 19.56 | 0.00102 | 37.7 |
| 256 | 0.000095 | 6.24 | 0.00037 | 24.4 |
| 384 | 0.000394 | 58.09 | 0.00107 | 157.3 |
| 512 | 0.000338 | 88.59 | 0.00202 | 528.3 |
| 768 | 0.006888 | 4062.71 | 0.01578 | 9310.2 |

N = #gaps, $\mathrm{err}(\mathbf{u}) := \max_j |\mathbf{u}_j - u(x_j)|$ , $\mathrm{err}(\mathbf{u}^{‡}) := \max_j |\mathbf{u}^{‡}_j - u'(x_j)|$ .

Different rounding errors cause this second program's computed $\mathbf{u}$ to differ insignificantly from the first's; again, discretization contributes error about $2.4/N^2$ , and roundoff is still amplified by a factor of the order of $N^2$ , so errors in $\mathbf{u}$ never get much below $\sqrt{\varepsilon}$ . Less roundoff occurs when N is a power of 2 because then divisions by N are exact. The second program computes the approximate gradient $\mathbf{u}^{‡}$ so much more accurately than the first did as to be adequate …

"… to give artistic versimilitude to an otherwise bald and unconvincing narrative"

from *The Mikado* by W.S. Gilbert and A.S. Sullivan

in a computerized game. For reliable scientific and engineering computation the uncertainty in $\mathbf{u}^{\ddagger}$, at least roughly $3.7/N^2 + 2N^3/10^{11}$, seems excessive. (The *uncertainty* is not the *error* but instead our least estimate of how big the error isn't; uncertainty is an error-bound.)

Like the first program, this second program loses at least about half the sig. bits carried by the arithmetic. A fourth program below will attenuate roundoff's amplification by appending suitably programmed iterative refinement to this second program.

**§10. Iterative Refinement:** Each iteration will replace the currently computed $\mathbf{u}$ by $\mathbf{u} + \Delta\mathbf{u}$ wherein $\Delta\mathbf{u}$ is the computed (and therefore approximate) solution of $E\cdot B\cdot\Delta\mathbf{u} = \mathbf{s}$ in which the residual $\mathbf{s}$ approximates $\mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$ to reveal the extent to which $\mathbf{u}$ dissatisfies the equation we wish to solve. The third program assigns "$\mathbf{s} := \mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$" computed just as it is written here in arithmetic rounded to the same precision as was used to compute $E$ and $B$.

But this program's iterative refinement never improves accuracies much and often worsens them when the second program's accuracies most need improvement. Such disappointing performance cannot come as a surprise when we recall that roundoff contaminates both the residual computed from the expression "$\mathbf{r} - (\mathbb{T} + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u}$" and the triangular factorization $\mathbb{T} + \mathrm{Diag}(\mathbf{q}) = E\cdot B$ about equally badly. Iterative refinement cannot be expected to improve the accuracy of $\mathbf{u}$ unless the residual $\mathbf{s}$ is computed more accurately than the factorization.

The fourth program differs from the third only by computing the residual $\mathbf{s}$ more accurately and, in benign computing environments, effortlessly. We assume all arrays $\mathbf{u}$, $\mathbf{u}^{\ddagger}$, $\mathbf{q}$, $\mathbf{r}$, $\{a_j\}$, $\{ß_j\}$ and $\{e_j\}$ to be stored as arrays of 4-byte `floats`. The elements $g_j := \mathbf{q}_j - a_{j-1} - a_j$ are assumed to be computed at the moments of need or else stored as an array of 8-byte `doubles`; trickier ways to cope with $\{g_j\}$ will be passed over for the sake of a simpler exposition.

In a benign computing environment, as was provided by the original Kernighan-Ritchie ***C*** and is now available fom some implementations of ***C***99, the `float` residual $\mathbf{s}$ can be computed amply accurately from any one of the three assignments

"$\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot\mathbf{u}_{j-1} - g_j\cdot\mathbf{u}_j - a_j\cdot\mathbf{u}_{j+1}$" or
"$\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot\mathbf{u}_{j-1} - (\mathbf{q}_j - a_{j-1} - a_j)\cdot\mathbf{u}_j - a_j\cdot\mathbf{u}_{j+1}$" or
"$\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot(\mathbf{u}_{j-1} - \mathbf{u}_j) - a_j\cdot(\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j\cdot\mathbf{u}_j$"

each of whose right-hand side's every arithmetic operation is rounded to `double` before being stored as a `float` in $\mathbf{s}_j$. Adequate accuracy is achieved thus with no extra effort nor thought.

In a FORTRANnish environment like JAVA or ANSII ***C*** (1987), the foregoing assignments must be encumbered by *casts* (conversions to `double`) to achieve amply adequate accuracy thus:

"$\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot(\texttt{double})u_{j-1} - g_j\cdot u_j - a_j\cdot(\texttt{double})u_{j+1}$" or
"$\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot(\texttt{double})u_{j-1} - (((\texttt{double})q_j - a_{j-1}) - a_j)\cdot u_j - a_j\cdot(\texttt{double})u_{j+1}$" or
"$\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot((\texttt{double})u_{j-1} - u_j) - a_j\cdot((\texttt{double})u_{j+1} - u_j) - q_j\cdot(\texttt{double})u_j$".

In a benighted environment where `double` is unavailable or too inconvenient or too slow, barely adequate accuracy can be achieved at the cost of two extra subtractions via a tricky expedient:

" $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot((\mathbf{u}_{j-1} - \mathbf{u}_j) - (\mathbf{u}_{j+1} - \mathbf{u}_j)) - (a_j - a_{j-1})\cdot(\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j\cdot\mathbf{u}_j$ " .
HONOR PARENTHESES !

This formula for residuals produced the results tabulated below, all computed using only `float` variables and precision throughout the fourth program. This formula should work in MATLAB 7. This formula is not preferable to the formulas above for benign or FORTRANnish environments, using `double` arithmetic upon `float` variables; those formulas produce slightly better results (not tabulated below) with no need for the analysis in §7 nor Appendix 1.

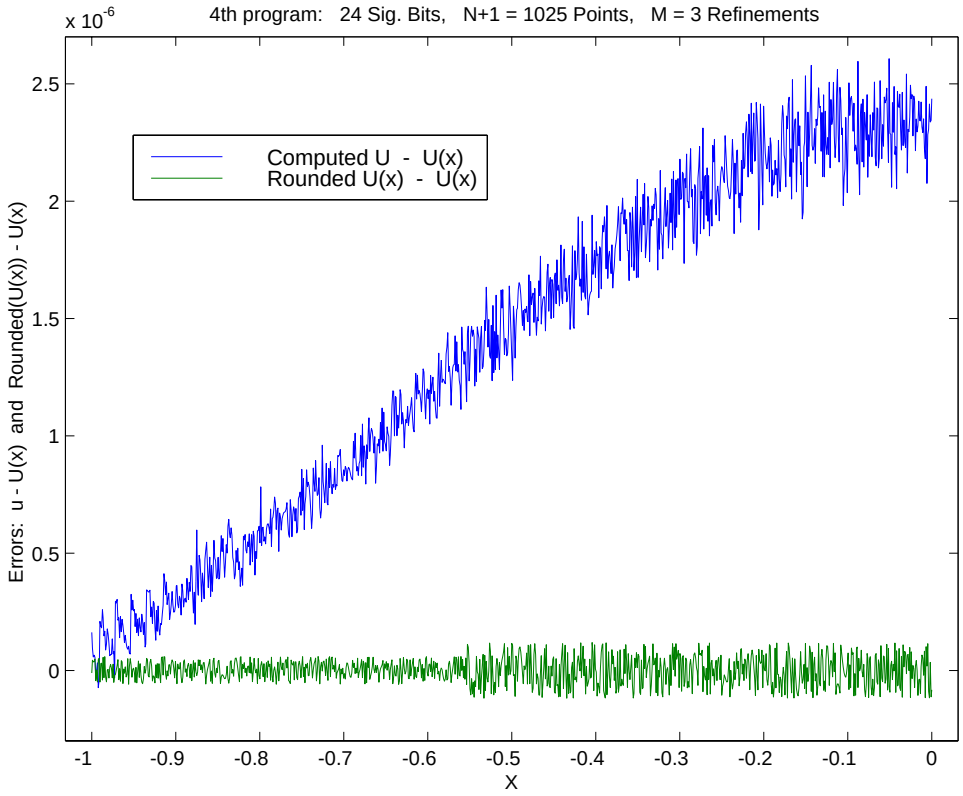
Results from the Fourth Program carrying 24 sig. bits ( $\varepsilon \approx 6/10^8$ )

| N | M | err(**u**) | err(**u**)·N$^2$ | err(**u**$^\ddagger$) | err(**u**$^\ddagger$)·N$^2$ | M | N |
|---|---|---|---|---|---|---|---|
| 16 | 0 & 1 | 0.00932 | 2.39 | 0.0153 | 3.9 | 0 & 1 | 16 |
| 24 | 0 & 1 | 0.00414 | 2.39 | 0.0066 | 3.8 | 0 & 1 | 24 |
| 32 | 0 & 1 | 0.002326 | 2.38 | 0.00365 | 3.7 | 0 & 1 | 32 |
| 48 | 0 | 0.001028 | 2.37 | 0.00158 | 3.6 | 0 | |
| | 1 & 2 | 0.0010349 | 2.38 | 0.001612 | 3.71 | 1 & 2 | 48 |
| 64 | 0 | 0.000663 | 2.73 | 0.00099 | 4.0 | 0 | |
| | 1 & 2 | 0.0005821 | 2.38 | 0.000904 | 3.70 | 1 & 2 | 64 |
| 96 | 0 | 0.000118 | 1.09 | 0.00022 | 2.0 | 0 | |
| | 1 & 2 | 0.0002586 | 2.38 | 0.000393 | 3.62 | 1 & 2 | 96 |
| 128 | 0 | 0.000073 | 1.19 | 0.00027 | 4.5 | 0 | |
| | 1 & 2 | 0.0001456 | 2.39 | 0.000206 | 3.38 | 1 & 2 | 128 |
| 192 | 0 | 0.000531 | 19.56 | 0.00102 | 37.7 | 0 | |
| | 1 & 2 | 0.0000646 | 2.38 | 0.000107 | 3.94 | 1 & 2 | 192 |
| 256 | 0 | 0.000095 | 6.24 | 0.00037 | 24.4 | 0 | |
| | 1 & 2 | 0.0000364 | 2.39 | 0.000061 | 4.00 | 1 & 2 | 256 |
| 384 | 0 | 0.000394 | 58.09 | 0.00107 | 157.3 | 0 | |
| | 1 | 0.0000162 | 2.39 | 0.000049 | 7.27 | 1 | |
| | 2 & 3 | 0.0000162 | 2.39 | 0.000053 | 7.81 | 2 & 3 | 384 |
| 512 | 0 | 0.000338 | 88.59 | 0.00202 | 528.3 | 0 | |
| | 1 | 0.0000091 | 2.38 | 0.000061 | 16.11 | 1 | |
| | 2 & 3 | 0.0000092 | 2.41 | 0.000065 | 16.92 | 2 & 3 | 512 |
| 768 | 0 | 0.006888 | 4062.71 | 0.01578 | 9310.2 | 0 | |
| | 1 | 0.0000156 | 9.20 | 0.000089 | 52.50 | 1 | |
| | 2 & 3 | 0.0000041 | 2.41 | 0.000088 | 51.75 | 2 & 3 | 768 |
| 1024 | 0 | 0.0037995 | 3984.06 | 0.012934 | 13561.9 | 0 | |
| | 1 | 0.0000062 | 6.45 | 0.000119 | 124.27 | 1 | |
| | 2 & 3 | 0.0000024 | 2.49 | 0.000114 | 119.36 | 2 & 3 | 1024 |
| 1536 | 0 | 0.0091051 | 21481.52 | 0.021961 | 531.6 | 0 | |
| | 1 | 0.0000313 | 73.85 | 0.000225 | 531.61 | 1 | |
| | 2 & 3 | 0.0000012 | 2.90 | 0.000174 | 411.38 | 2 & 3 | 1536 |
| 2048 | 0 | 0.0381667 | 160082.62 | 0.076282 | 319949.1 | 0 | |
| | 1 | 0.0004934 | 2069.63 | 0.001069 | 4483.10 | 1 | |
| | 2 | 0.0000057 | 23.79 | 0.000240 | 1006.40 | 2 | |
| | 3 | 0.0000006 | 2.56 | 0.000243 | 1019.33 | 3 | |
| | 4 & 5 | 0.0000007 | 2.86 | 0.000236 | 988.87 | 3 & 4 | 2048 |

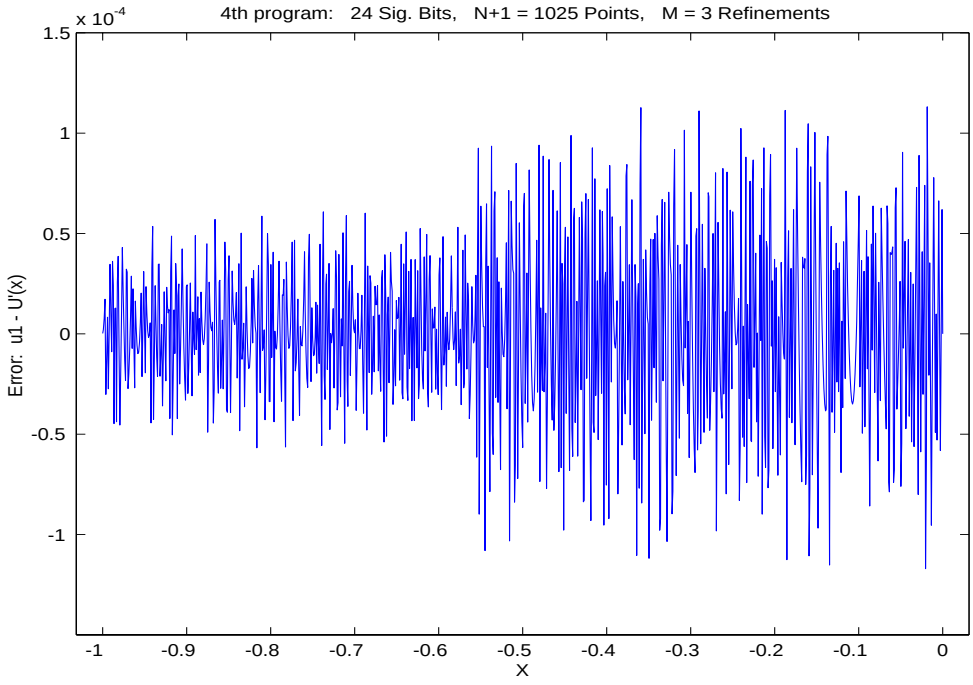
N = #gaps, M = #refinements, err(**u**) := $\max_j |\mathbf{u}_j - u(x_j)|$ , err(**u**$^\ddagger$) := $\max_j |\mathbf{u}^\ddagger_j - u'(x_j)|$ .

The dwindling accuracy of the estimated gradient $\mathbf{u}^{\ddagger}$ as N increases beyond about 500 evinces a *Law of Diminishing Returns* enforced by roundoff. How? The graphs below explain it all:

Error in the Fourth Program's $\mathbf{u} \approx u(x)$ computed carrying 24 sig. bits, N = 1024, M = 3




Error in the Fourth Program's $\mathbf{u}^{\ddagger} \approx u'(x)$ computed carrying 24 sig. bits, N = 1024, M = 3



The dwindling accuracy of the estimated gradient $\mathbf{u}^{\ddagger}$ as N increases beyond about 500 comes mostly from the act of rounding arrays $\{x_j\}$, $\{a_j\}$ and $\mathbf{q}$ and computed solution $\mathbf{u}$ to `float`'s 24 sig. bits. Roundoff adds raggedness to an otherwise smooth graph of $\mathbf{u}$'s discretization error which grows to about $2.36/N^2$. Much of $\mathbf{u}$'s raggedness comes from rounding $\mathbf{u}$ and doubles from about $\varepsilon$ to $2\varepsilon = 2^{-23} \approx 1.2/10^7$ as x increases past –0.553943 and $u(x)$ increases past 2. Then computing $\mathbf{u}^{\ddagger}$ from the divided differences of $\mathbf{u}$ amplifies its raggedness by a factor often almost as big as N. The graphs corroborate these estimates; and the the tabulated results corroborate error-analyses that predict $\mathbf{u}^{\ddagger}$'s loss of at least roughly a third of the arithmetic's sig. digits if the gradient is computed from the divided-difference quotients used in our four programs. Two thirds of `float`'s 24 sig. bits is accuracy adequate for most engineering applications of $\mathbf{u}^{\ddagger}$, and this much accuracy is achieved by using the tricks presented in these notes.

The fourth program's tabulated results reveal another way for an excessively big N to exacerbate the effect of roundoff: It slows the convergence of iterative refinement. This happens because the condition number $\kappa(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) := \|\mathbb{T} + \mathrm{Diag}(\mathbf{q})\| \cdot \|(\mathbb{T} + \mathrm{Diag}(\mathbf{q}))^{-1}\|$ tcomes to roughly $4N^2$ for any plausible norm $\|\ldots\|$. Iterative refinement converges quickly only if roundoff disturbs the triangular factorization of $\mathbb{T} + \mathrm{Diag}(\mathbf{q})$ by rather less than its distance from the nearest singular matrix, and this happens only if $\varepsilon \cdot \kappa(\mathbb{T} + \mathrm{Diag}(\mathbf{q})) << 1$. This implies that convergence is too likely to go slow unless $N << 1/\sqrt{4\varepsilon} = 2048$. The bound is corroborated by the tabulated results, which exhibit accuracy adequate for most practical purposes achieved by one refinement at gap-counts N roughly between 100 and 500, and more refinements needed when $N \geq 1536$.

There are better ways to solve the foregoing example's boundary-value problem. *Shooting methods* that recast the boundary-value problem as a sequence of initial-value problems work well with this example, if shooting starts at the differential equation's singularity, because the recast differential equation is stable; $u(x) = v(x)/v(1)$ from §3. But no such methods work upon the partial differential equations for which this note's trick is intended. Another method that works well upon the example is *Collocation of Splines*; this method is implemented as `bvp4c` in recent MATLAB versions. Analogous methods for partial differential equations are too complicated to get used much.

**§11: Discretization of an Elliptic Boundary-Value Problem:** Given $\Phi(x,y)$ on the boundary $\partial\Omega$ of the unit square $[0, 0] \leq [x, y] \leq [1, 1]$, we seek the solution $\Phi(x,y)$ inside $\Omega$ of

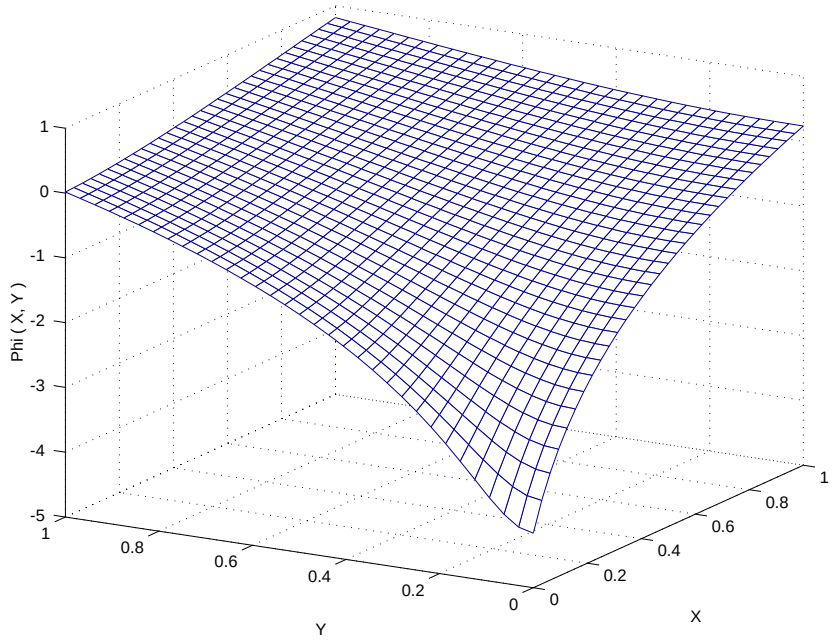
*Laplace's Equation*: $\mathrm{Div}\,\mathrm{Grad}\,\Phi(x,y) = \nabla^2\Phi(x,y) := \partial^2\Phi(x,y)/\partial^2 x + \partial^2\Phi(x,y)/\partial^2 y = 0$.

Numerical solutions $F(x,y)$ will be compared with $\Phi(x,y) := \log((x+\frac{1}{8})^2 + y^2)$ whose boundary values on $\partial\Omega$ have been chosen for this example. F will approximate $\Phi$ inside $\Omega$ at the $(N-1)^2$ intersections of a mesh that covers $\Omega$ by small squares each $\theta := 1/N$ on a side. On this mesh the differential operator $\nabla^2$ is approximated by a difference operator $\spadesuit$ defined thus:

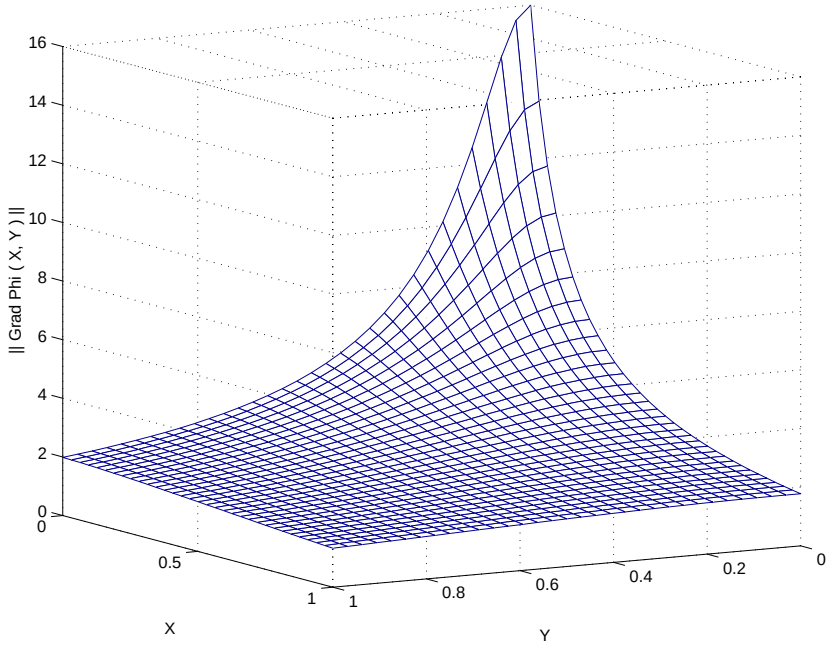
$$\spadesuit\Phi(x,y) := \big( \Phi(x-\theta,y) + \Phi(x,y-\theta) - 4\cdot\Phi(x,y) + \Phi(x+\theta,y) + \Phi(x,y+\theta) \big)/\theta^2$$
$$= \nabla^2\Phi(x,y) + O(\theta^2) \text{ as } \theta \to 0 .$$

Discretization approximates $\Phi$ by the solution F of $\spadesuit F = 0$ on the mesh inside $\Omega$. This last equation turns into a system $A\cdot\boldsymbol{f} = \mathbf{b}$ of linear equations in which column $\boldsymbol{f}$ contains the values of F inside $\Omega$, and column $\mathbf{b}$ is determined by $4\cdot(N-1)$ values on $\partial\Omega$. These columns and matrix A have huge dimension $(N-1)^2$ but A is sparse with bandwidth $2N-1$.

$$-4.16 < \Phi(x, y) = \log( (x+1/8)^2 + y^2 ) < 0.82$$




$$1.3 < \| \text{ Grad } \Phi(x, y) \| \le 16$$




The coordinates' origin is behind and under this surface.

Grad $\Phi(x, y)$ is the transpose of $\Phi'(x, y) = 2\cdot[x+\frac{1}{8},\ y]/((x+\frac{1}{8})^2 + y^2)$, to be approximated by

$$\Phi^{\ddagger}(x,y) := [\Phi(x+\theta,y)-\Phi(x-\theta,y),\ \Phi(x,y+\theta)-\Phi(x,y-\theta)]/(2\theta) \ = \ \Phi'(x,y) + O(\theta^2).$$

Equation "$A\cdot\boldsymbol{f} = \mathbf{b}$" was solved by *Successive Over-Relaxation* to get a first approximation $\mathbf{f}$ to the column $\boldsymbol{f}$ of values of F. This iteration's stopping criterion was chosen to avoid *dithering*; see my web pages' …/Math128/SlowIter.pdf . Then one iterative refinement computed residual $\mathbf{r} := A\cdot\mathbf{f} - \mathbf{b}$ and solved "$A\cdot\Delta\mathbf{f} = \mathbf{r}$" to approximate $\boldsymbol{f}$ better, presumably, by $\mathbf{f} - \Delta\mathbf{f}$.

The foregoing process was performed thrice, first using an ordinary matrix multiply to compute $A\cdot\mathbf{f}$ and $A\cdot\Delta\mathbf{f}$, and second computing them with a trick that replaced the crude formula

$$\spadesuit F(x,y) := \big( F(x-\theta,y) + F(x,y-\theta) - 4\cdot F(x,y) + F(x+\theta,y) + F(x,y+\theta) \big)/\theta^2$$

by the algebraically equivalent but numerically more accurate 2nd-order ♠ formula

$$\spadesuit F(x,y) := \big( \big((F(x+\theta,y) - F(x,y)) - (F(x,y) - F(x-\theta,y))\big) + \big((F(x,y+\theta) - F(x,y)) - (F(x,y) - F(x,y-\theta))\big) \big)/\theta^2$$

to take advantage of exact subtractive cancellations. These two computations were performed entirely in 4-byte `float` arithmetic carrying 24 sig.bits. The third computation of F was performed in 8-byte `double` carrying 53 sig.bits to nearly nullify roundoff. Thus, six sets of values F and their errors $E := \max_{x,y}|F(x,y) - \Phi(x,y)|$ were generated to be compared:

| | |
|---|---|
| $E_{true2}$ | E computed from the tricky formula for ♠F carrying 53 sig.bits. |
| $E_{trick2}$ | E computed from the tricky formula for ♠F carrying 24 sig.bits. |
| $E_{crude2}$ | E computed from the crude formula for ♠F carrying 24 sig.bits. |

Each E was recorded both before and after iterative refinement.

Everything above was repeated for each of a sequence of diminishing mesh-gaps $\theta = 1/N$. Since $\spadesuit\Phi = \nabla^2\Phi + O(\theta^2)$ we expected $E\cdot N^2$ to approach a constant as $N \to \infty$ except for roundoff.

Tabulated below are computed results with E before iterative refinement shown above E after:

| N | $E_{true2}$ | …·$N^2$ | $E_{trick2}$ | …·$N^2$ | $E_{crude2}$ | …·$N^2$ |
|---|---|---|---|---|---|---|
| 128 | 7.481e-5 | 1.226 | 2.030e-4 | 3.326 | 2.025e-4 | 3.318 |
| | | | 7.472e-5 | 1.224 | 7.816e-5 | 1.281 |
| 256 | 1.872e-5 | 1.227 | 7.440e-5 | 4.876 | 7.766e-5 | 5.089 |
| | | | 1.879e-5 | 1.231 | 5.081e-5 | 3.33 |
| 512 | 4.681e-6 | 1.227 | 1.878e-5 | 4.924 | 5.076e-5 | 13.31 |
| | | | 4.787e-6 | 1.255 | 4.434e-5 | 11.62 |
| 1024 | 1.170e-6 | 1.227 | 4.912e-6 | 5.151 | 4.440e-5 | 46.56 |
| | | | 1.285e-6 | 1.348 | 3.363e-5 | 35.27 |
| 2048 | 2.926e-7 | 1.227 | 1.488e-6 | 6.240 | 3.386e-5 | 142 |
| | | | 4.085e-7 | 1.713 | 3.394e-5 | 142.4 |

Worse!

The crude formula for ♠ lost almost two sig.dec. more than the tricky formula, which allowed iterative refinement to render the final error E almost as small as if F matched Φ to 23 sig.bits. The crude formula's residual was not accurate enough to ensure that iterative refinement would always diminish the error $E$.

**§12. Computing Grad Φ :** When derivative $\Phi'$, approximated by the

*Central Divided Difference* $\Phi^{\ddagger}(x,y) := [\Phi(x+\theta,y)-\Phi(x-\theta,y),\ \Phi(x,y+\theta)-\Phi(x,y-\theta)]/(2\theta)$
$= \Phi'(x,y) + O(\theta^2)$,

is further approximated by the computed $F^{\ddagger}(x, y)$, sources of error accrue to include …

- $O(\theta^2)$ inherited from $\Phi^{\ddagger}$, and
- error $F^{\ddagger} - \Phi^{\ddagger} = (F - \Phi)^{\ddagger}$ due to the differential equation's discretization, and
- at least $O(\varepsilon\cdot F/\theta)$ due to roundoff's contamination of F.

The relative magnitudes of these sources are not often knowable in advance. For instance, the second source $F^{\ddagger} - \Phi^{\ddagger}$ is usually much smaller than $(F - \Phi)/\theta$ because the discretization's error $F - \Phi$ is usually smoothly *Pillow-Shaped*, as was the error (Computed **u** – *u*) plotted on p.18 (after its raggedness due to roundoff is smoothed away). The third source's rounding errors depend upon the numerical method in detail including any trick intended to attenuate them.

The first source's $O(\theta^2)$ turned out to be the preponderant contributor to our example's error $F^{\ddagger} - \Phi'$ at gap sizes θ whose $F - \Phi$ was about as small as roundoff allowed after tricks. This became evident when F was is recomputed using a higher-order (6th) discretization that reduced $F - \Phi$ from $O(\theta^2)$ to $O(\theta^6)$ for θ small enough though perhaps not so small as before. The higher-order discretization replaced the discretized Laplacian $\spadesuit\Phi(x,y)$ everywhere above by …

$$\begin{aligned}\clubsuit\Phi(x,y) := (\ & \Phi(x-\theta,y+\theta) + 4\cdot\Phi(x,y+\theta) + \Phi(x+\theta,y+\theta) + \\ & 4\cdot\Phi(x-\theta,y) - 20\cdot\Phi(x,y) + 4\cdot\Phi(x+\theta,y) + \\ & \Phi(x-\theta,y-\theta) + 4\cdot\Phi(x,y-\theta) + \Phi(x+\theta,y-\theta)\ )/(6\cdot\theta^2)\end{aligned}$$
$$= \nabla^2\Phi(x,y) + \nabla^4\Phi(x,y)\cdot\theta^2/12 + (\ \nabla^6\Phi(x,y) + 2\partial^4\nabla^2\Phi(x,y)/\partial x^2\partial y^2\ )\cdot\theta^4/360 + O(\theta^6)$$
$$= O(\theta^6) \text{ if } \nabla^2\Phi = 0 .$$

The trick that attenuated most of the roundoff in ♣F computed it from …

$$\clubsuit F(x,y) := (\ 4\cdot(\theta^2\cdot\spadesuit F(x,y)) + (((F(x-\theta,y+\theta) - F(x,y)) + (F(x+\theta,y-\theta) - F(x,y))) + ((F(x-\theta,y-\theta) - F(x,y)) + (F(x+\theta,y+\theta) - F(x,y))))\ )/(6\cdot\theta^2) .$$

As before, six sets of values F and their errors $E := \max_{x,y}|F(x,y) - \Phi(x,y)|$ were generated, but now from the equation "♣F(x,y) := 0", to be compared:

| | |
|---|---|
| $E_{true6}$ | E computed from the tricky formula for ♣F carrying 53 sig.bits. |
| $E_{trick6}$ | E computed from the tricky formula for ♣F carrying 24 sig.bits. |
| $E_{crude6}$ | E computed from the crude formula for ♣F carrying 24 sig.bits. |

Each E is tabulated below both before (above) and after iterative refinement (under). Also tabulated is $E_{true2}$ to facilitate a comparison of accuracies from ideal 2nd-order ♠ and 6th-order ♣ formulas unobscured by roundoff, though it crept into ♣ at the bottom of the table.

The crude ♣ lost about one sig.dec. more than the tricky ♣, whose iterative refinement for $N \geq 64$ produced F matching Φ to about 23 sig.bits. The crude formula's residual was not accurate enough to ensure that iterative refinement would always diminish the error *E*.

Errors $E := \max_{x,y}|F(x,y) - \Phi(x,y)|$

| N | $E_{true6}$ | $\ldots\cdot N^6$ | $E_{trick6}$ | $E_{crude6}$ | $E_{true2}$ |
|---|---|---|---|---|---|
| 16 | 9.677e-5 | 1.6e3 | 9.681e-5 | 9.705e-5 | 4.387e-3 |
| | | | 9.681e-5 | 9.658e-5 | |
| 32 | 2.084e-6 | 2.2e3 | 2.039e-6 | 2.039e-6 | 1.179e-3 |
| | | | 2.039e-6 | 2.039e-6 | |
| 64 | 3.225e-8 | 2.2e3 | 6.928e-7 | 9.254e-6 | 2.979e-4 |
| | | | 1.708e-7 | 1.635e-6 | |
| 128 | 5.126e-10 | 2.3e3 | 1.859e-6 | 2.760e-6 | 7.481e-5 |
| | | | 2.845e-7 | 2.290e-6 | |
| 256 | 8.103e-12 | 2.3e3 | 5.362e-7 | 2.366e-6 | 1.872e-5 |
| | 8.104e-12 | 2.3e3 | 2.283e-7 | 2.720e-6 | |
| 512 | 1.121e-13 | 2.0e3 | 3.665e-7 | 2.685e-6 | 4.681e-6 |
| | 1.266e-13 | 2.3e3 | 2.900e-7 | 8.151e-6 | |

An adequately accurate approximation to the gradient required gap-sizes $\theta = 1/N$ rather smaller than sufficed for adequate accuracy in $F$. The error $D := \max_{x,y}||F^{\ddagger}(x,y) - \Phi'(x,y)||$ was found for both crude and tricky versions of both ♠ and ♣, for arithmetics carrying both 24 and 53 sig.bits, and before and after iterative refinement though it made little difference. The following versions of $D$ are tabulated below:

$D_{true6}$ $D$ computed from the tricky formula for ♣F carrying 53 sig.bits.
$D_{trick6}$ $D$ computed from the tricky formula for ♣F carrying 24 sig.bits.
$D_{crude6}$ $D$ computed from the crude formula for ♣F carrying 24 sig.bits.
$D_{trick2}$ $D$ computed from the tricky formula for ♠F carrying 24 sig.bits.

$D := \max_{x,y}||F^{\ddagger}(x,y) - \Phi'(x,y)||$

| N | $D_{true6}$ | $\ldots\cdot N^2$ | $D_{trick6}$ | $D_{crude6}$ | $D_{trick2}$ |
|---|---|---|---|---|---|
| 16 | 0.3207 | 82.09 | 0.3207 | 0.3207 | 0.2948 |
| | | | 0.3207 | 0.3207 | 0.2948 |
| 32 | 0.161 | 164.8 | 0.161 | 0.161 | 0.1537 |
| | | | 0.161 | 0.161 | 0.1537 |
| 64 | 0.05773 | 236.5 | 0.05772 | 0.05767 | 0.05467 |
| | | | 0.05773 | 0.05772 | 0.05458 |
| 128 | 0.01731 | 283.6 | 0.01732 | 0.0174 | 0.01593 |
| | | | 0.01731 | 0.01732 | 0.0161 |
| 256 | 0.004745 | 311 | 0.004778 | 0.004847 | 0.004456 |
| | | | 0.004727 | 0.004847 | 0.004415 |
| 512 | 0.001243 | 325.8 | 0.001297 | 0.001398 | 0.001328 |
| | | | 0.001297 | 0.001398 | 0.001231 |
| 1024 | 0.000318 | 333.4 | ---- | ---- | 0.000526 |
| | | | ---- | ---- | 0.0004997 |
| 2048 | 8.043e-5 | 337.4 | ---- | ---- | 0.0008799 |
| | | | ---- | ---- | 0.0006271 |

The tabulated magnitudes of the errors $F^{\ddagger} - \Phi'$ reflect a contribution roughly $333\cdot\theta^2$ due mostly to either a mesh-gap $\theta$ too big, or a 2nd-order formula $F^{\ddagger}$ too crude, rather than $F - \Phi$ too inaccurate. This could hardly have been known in advance. 4th-order formulas more refined than $F^{\ddagger}$ come from the calculus of divided differences as follows:

Given a sufficiently differentiable $f(x)$, its derivative $f'(x)$ is approximated by

$$f^{\dagger}(x,\theta) := ( f(x+\theta) - f(x) )/\theta = f'(x) + O(\theta) ;$$
$$f^{\ddagger}(x,\theta) := ( f^{\dagger}(x+\theta) + f^{\dagger}(x-\theta) )/2 = f'(x) + O(\theta^2) ;$$

$$( 4\cdot f^{\ddagger}(x,\theta) - f^{\ddagger}(x,2\theta) )/3 = f'(x) + O(\theta^4) ;$$
$$4\cdot f^{\dagger}(x,\theta) - 6\cdot f^{\dagger}(x,2\theta) + 4\cdot f^{\dagger}(x,3\theta) - f^{\dagger}(x,4\theta) = f'(x) + O(\theta^4) ;$$
$$f^{\dagger}(x,-\theta)/4 + 3\cdot f^{\dagger}(x,\theta)/2 - f^{\dagger}(x,2\theta) + f^{\dagger}(x,3\theta)/4 = f'(x) + O(\theta^4) .$$

The last three formulas would take better advantage of the accuracy of $F$ computed from ♣, but at the cost of complication: One formula works better than the others deep inside the square $\Omega$, and the last two are needed near its boundary. The complication does not alter our tricks.

**§13. Conclusions:** These notes' thesis, supported by analysis and examples, is that boundary-value problems $\mathrm{div}(\boldsymbol{p}\cdot\mathbf{grad}\, u) + q\cdot u = r$ can be discretized by divided differences and solved faster and accurately enough for most practical purposes (including gradients) when all arrays are stored as 4-byte `floats` instead of `doubles` 8 bytes wide, and all arithmetic is rounded to `floats`' 24 sig. bits, as many current graphics processors do. Residuals must be computed accurately enough via a trick effortless only if cancelling residuals are computed using arithmetic rounded to higher precision, say the 53 sig. bits of `double` precision, which computers used to do automatically when programmed in Kernighan-Ritchie ***C*** as it was before the mid-1980s.

The examples were chosen to illustrate three additional observations:

• Without tricks, `floats` are now too inaccurate for reliable scientific and engineering work.

• Rounding errors can corrupt severely a regular solution of a singular differential equation unless the discretization is designed to filter out singular solutions and also to preserve vital symmetries.

• If residuals are computed well enough, the accuracy of a computed solution tends to improve with iterative refinement after the discretization is refined by an increase in the density of mesh-points. But the rate of improvement declines as a solution's accuracy approaches the arithmetic's precision; and further mesh refinement incurs retardation of iterative refinement's convergence.

The tricks presented in these notes are palliatives, not cures for ailments that afflict scientific and engineering computation in an era when floating-point arithmetic is employed overwhelmingly more often for games and entertainment. Two of the ailments are, first, a lack of programming tools to help diagnose failure modes peculiar to floating-point computation and, second, wide-spread misunderstandings of roundoff among scientists, engineers and even numerical analysts. Because almost all of them view cancellation as an enemy rather than an ally to combat roundoff, they are predisposed to overlook the trickier tricks in these notes. Education will not cure their misunderstandings since a study of roundoff is so unlikely to be added to an already overloaded college undergraduate's syllabus. Besides, students forget tricks taught but not soon exercised.

Incurable ailments are best prevented by prophylaxis like vaccination, healthy diet and exercise, and seat-belts and air-bags. The analogous prophylaxis for numerical computation is arithmetic extravagantly more precise than the data and the accuracy desired in results. Ideally programming languages should supply this much precision *by default*, without requiring an explicit request from programmers naive about floating-point roundoff though clever at things they care about. If a daredevil programmer chooses to trade accuracy away for speed, let that be *his* decision, not decided by the designers and implementors of programming languages and program development environments, nor a decision forced upon them by obeisance to benchmarks that rate only speed regardless of reliability. Languages for typical programmers should presuppose this *mantra*:

> Routinely (by default) perform *all* arithmetic and carry *all* intermediate variables extravagantly more precisely than the data and the accuracy desired in computed results.

It's just a matter of time until every one of us has occasion to depend upon software promulgated perhaps over the Internet and produced by some programmer numerically naive but otherwise clever, maybe ourself. The interests of all of us are served better if programming environments are designed first to help get things right and after that, if need be, to help speed them up.

**§14. Appendix 1:** This concerns a trick to compute scalar $z := A\cdot x - B\cdot y$ more accurately when it cancels severely enough that $|z| << Z := |A\cdot x| + |B\cdot y|$ because both $A/B$ and $x/y$ are so near 1. If $z$ is computed naively from the expression "$A\cdot x - B\cdot y$" literally, its uncertainty due to the two multiplications' roundoff will be of the order of $\pm\varepsilon\cdot Z$, where $\varepsilon$ is the roundoff threshold. But whenever both $1/2 \le e^{-2\theta} \le A/B \le e^{2\theta} \le 2$ and $1/2 \le e^{-2\phi} \le x/y \le e^{2\phi} \le 2$, the trick reduces the uncertainty in $z$ to something of the order of $\pm\varepsilon\cdot(|z| + Z\cdot\tanh(\theta+\phi))$. This is substantially smaller than $\pm\varepsilon\cdot Z$ when $\theta$ and $\phi$ are both tiny of order $h_{...}$, as is the case when the trick is used to compute $a_{j-1}\cdot(\mathbf{u}_{j-1} - \mathbf{u}_j) - c_{j+1}\cdot(\mathbf{u}_j - \mathbf{u}_{j+1})$ more accurately for tiny gaps $h_{j-1}$ and $h_j$.

Before the trick is explained it will be liberated from a spurious argument that would render the trick superfluous. The argument presumes that at least some of the data has inherited uncertainty from previous computation; say $A = a\cdot(1 \pm \varepsilon)$ and $B = b\cdot(1 \pm \varepsilon)$ because roundoff has altered their computed values away from their ideal but now unknown values $a$ and $b$ resp. Then even if no further rounding error occurred the computed value of $z = a\cdot x - b\cdot y \pm \varepsilon\cdot(|a\cdot x| + |b\cdot y|)$ would inherit uncertainty $\pm\varepsilon\cdot(|a\cdot x| + |b\cdot y|)$ almost the same as the uncertainty $\pm\varepsilon\cdot(|A\cdot x| + |B\cdot y|)$ that the trick is designed to attenuate. Thus the trick could get rid of at most about half the uncertainty that roundoff adds to $z$. If this argument were correct, the trick would not be worthwhile.

The argument would be correct if $z$ were the only thing computed from the data $A$, $B$, $x$ and $y$. The argument's logic falls short when some of this data appears in other expressions like $z$ and destined to combine with it. The argument overlooks the fact that uncertainties due to roundoff are not uncorrelated, much less random. To succeed, error-analysis must take correlations into account lest its excessive pessimism generate misconceived advice and bad decisions.

The trick computes $z$ not from the expression "$A\cdot x - B\cdot y$" but from either of two formulas

"$z := (A-B)\cdot x + B\cdot(x-y)$" and "$z := A\cdot(x-y) + (A-B)\cdot y$"

suggested by the *Calculus of Divided Differences*; see the product rule on p. 2 of my posting

<www.cs.berkeley.edu/~wkahan/Math185/Derivatives.pdf>.

The hypothesis that both $1/2 \le A/B \le 2$ and $1/2 \le x/y \le 2$ ensures that "$x-y$" and "$A-B$" are computed *exactly* despite cancellation, at least if the arithmetic conforms to IEEE Standard 754 for Binary Floating-Point Arithmetic, and also for practically all current hardware arithmetics.

There were hardware arithmetics designed in the 1960s, some lingering into the early 1990s, whose subtractions lacked a *Guard Digit* and consequently could not guarantee substantial cancellation free from new roundoff. Such arithmetics died after hardware designers learned that lack of a guard digit conferred no performance advantage. Young designers of new fast graphics processors may not yet have learned that lesson. A guard digit is usually omitted from software-simulated floating-point whose precision exceeds what is built into the hardware, but such arithmetic is unlikely to figure in the circumstances pertinent here.

Let's choose the first of the two formulas above. Its value actually computed for $z$ is

$$z \pm \varepsilon\cdot(\zeta + |z|) := A\cdot x - B\cdot y \pm \varepsilon\cdot(|A-B|\cdot|x| + |B|\cdot|x-y| + |z|)$$

when rounding errors in the two multiplications and the addition are taken into account. Thus, to compare the trick's $z \pm \varepsilon\cdot(\zeta + |z|)$ with the naively computed $z \pm \varepsilon\cdot Z$ we must compare the magnitudes of …

$$z := A\cdot x - B\cdot y, \quad Z := |A\cdot x| + |B\cdot y| \quad \text{and} \quad \zeta := |A-B|\cdot|x| + |B|\cdot|x-y|$$

when $e^{-2\theta} \le A/B \le e^{2\theta}$ and $e^{-2\phi} \le x/y \le e^{2\phi}$, and $\theta$ and $\phi$ are both tiny.

To simplify the comparisons without loss of generality we assume first that $0 < B \le A \le e^{2\theta}\cdot B$; otherwise we first reverse the signs of z, A and B and/or swap A with B and x with y. The second simplification reverses the signs of x, y and z if necessary to reduce the comparisons to two cases: either $0 < y \le x \le e^{2\phi}\cdot y$ or else $0 < x \le y \le e^{2\phi}\cdot x$. In both cases we find that

$$|z|/Z \le \tanh(\theta+\phi)\,, \quad \zeta/Z \le \tanh(\theta+\phi)\,, \text{ and } \quad (\zeta + |z|)/Z \le 2\cdot\tanh(\theta+\phi)\,.$$

The verification of these inequalities is left to the very diligent reader.

Consequently, as $\theta \to 0$ and $\phi \to 0$ independently, the ratio $(\zeta + |z|)/Z$ of the trick's to the naive evaluation's uncertainties tends to zero at least as fast as $2(\theta+\phi)$ does.

Whether this tricky attenuation of uncertainty due to roundoff is worth its cost in extra arithmetic depends first upon how many digits of accuracy you can afford to lose, and second upon your programming skill at overlapping and pipelining the two extra arithmetic operations. No such trick nor other artifice is needed for programs in languages that, on computers most widespread atop laps and under desks, can evaluate every arithmetic expression extra-precisely by default.

**§15. Appendix 2:** A divided-difference discretization that converges at higher order (faster) as all gaps approach zero need not necessarily entail higher order divided differences which increase the bandwidth or otherwise degrade the sparsity of the discretized equations that must be solved.

The discretization of the boundary-value problem $(P\cdot U')' + Q\cdot U = R$ poses a dilemma when the solution $U(x)$ fluctuates so much faster over some of its domain than over the rest as to call for a grid with varying gaps $h_j$ smaller where $U(x)$ fluctuates faster. Which of the following three strategies should be tried first?

- Keep all the gaps $h_j := h$ the same and use a simple finite-difference formula whose discretization error is of order $h^2$ or smaller though the density of mesh-points will be excessive wherever the solution fluctuates languidly.
- Vary the gaps $h_j$ *Adaptively* (making them smaller wherever $U(x)$ fluctuates faster) and use a divided-difference formula like that in §4 whose discretization error of low order $|h_j - h_{j-1}| + (h_j + h_{j-1})^2$ requires grid points of higher density wherever it varies.
- Vary the gaps $h_j$ adaptively using a complicated divided-difference formula like $\pounds U$ below whose discretization error of order $h_j^2 + h_{j-1}^2$ allows a lower mesh-point density while preserving the tridiagonal form of the discretized equations to be solved for **u**.

To exhibit that complicated formula $\pounds U$ is the purpose of this appendix. $\pounds U$ approximates not the differential operator $\mathbb{L}U(x) := P(x)\cdot U''(x) + P'(x)\cdot U'(x) = \big(P(x)\cdot U'(x)\big)'$ but a composite

$$\mathbb{L}U(x) + (1/3)(h-k)\cdot\big(\,\mathbb{L}U(x+h) - \mathbb{L}U(x-k)\,\big)/(h+k)$$

wherein $h = h_j$ and $k = h_{j-1}$ are the small gaps immediately astride $x = x_j$. The approximation is intended to be substituted not into the given differential equation $\mathbb{L}U(x) + Q(x)\cdot U(x) = R(x)$ but into its algebraically equivalent reformulation …

$$\begin{aligned}&\mathbb{L}U(x) + (1/3)(h–k)\cdot\big(\ \mathbb{L}U(x+h) – \ \mathbb{L}U(x–k)\ \big)/(h+k)\\ &\quad + Q(x)\cdot U(x) + (1/3)(h–k)\cdot\big(\ Q(x+h)\cdot U(x+h) – Q(x–k)\cdot U(x–k)\ \big)/(h+k)\\ &\qquad = R(x) + (1/3)(h–k)\cdot\big(\ R(x+h) – R(x–k)\ \big)/(h+k)\end{aligned}$$

at each internal mesh-point $x = x_j$ .

The complicated discrete approximation $\mathbb{£}U(x, h, k)$ to the composite differential/difference operator above is built out of several difference operators thus:

$$U^{\dagger}(x, h) := \big(\ U(x+h) – U(x)\ \big)/h\ ; \qquad P^{\dagger}(x, h) := \big(\ P(x+h) – P(x)\ \big)/h\ ;$$
$$\mathbb{¥}U(x, h) := \big(\ P(x+h) + P(x)\ \big)\cdot U^{\dagger}(x, h)\ ;$$
$$\mathbb{\$}U(x, h, k) := \big(\ U^{\dagger}(x, h) – U^{\dagger}(x, –k)\ \big)\cdot P^{\dagger}(x–k, h+k) – \big(\ P^{\dagger}(x, h) – P^{\dagger}(x, –k)\ \big)\cdot U^{\dagger}(x–k, h+k)\ ;$$
$$\mathbb{£}U(x, h, k) := \Big(\big(\ \mathbb{¥}U(x, h) – \mathbb{¥}U(x, –k)\ \big) + (1/3)\cdot(h–k)\cdot\mathbb{\$}U(x, h, k)\Big)/(h+k)\ .$$

The following substitutions discretize the reformulated differential equation: Replace it by

$$\begin{aligned}&\mathbb{£}U(x, h, k) + Q(x)\cdot U(x) + (1/3)(h–k)\cdot\big(\ Q(x+h)\cdot U(x+h) – Q(x–k)\cdot U(x–k)\ \big)/(h+k)\\ &\qquad = R(x) + (1/3)(h–k)\cdot\big(\ R(x+h) – R(x–k)\ \big)/(h+k)\end{aligned}$$

and then substitute $x_j$ for $x$ , $h_j$ for $h$ , $h_{j–1}$ for $k$ , and $\mathbf{u}_j$ for $U(x_j)$ at every internal mesh-point $x_j$ to produce a tridiagonal system of linear equations to be solved for the column $\mathbf{u}$ . Its discretization error turns out to be of second-order in the gap sizes because

$$\mathbb{L}U(x) + (1/3)(h–k)\cdot\big(\ \mathbb{L}U(x+h) – \ \mathbb{L}U(x–k)\ \big)/(h+k)\ – \ \mathbb{£}U(x, h, k)\ = \ O(h^2 + k^2)\ .$$

If $U'(x)$ appears in a boundary condition its discretization can be obtained from the formula

$$\big(\ k\cdot U^{\dagger}(x, h) – h\cdot U^{\dagger}(x, k)\ \big)/(k–h) = U'(x) + O(h\cdot k)\ .$$

If tricky suppression of roundoff in $\mathbb{£}U(x, h, k)$ is needed, restrict gaps $h_j$ to powers of 1/2 as explained in §7, and change the expression "$\mathbb{¥}U(x, h) – \mathbb{¥}U(x, –k)$" in the definition above of $\mathbb{£}U(x, h, k)$ to "$\big(\ P(x+h) – P(x–k)\ \big)\cdot U^{\dagger}(x, h) + \big(\ P(x–k) + P(x)\ \big)\cdot\big(\ U^{\dagger}(x, h) – U^{\dagger}(x, –k)\ \big)$" when computing residuals. Another small improvement replaces "$Q(x+h)\cdot U(x+h) – Q(x–k)\cdot U(x–k)$" by "$Q(x+h)\cdot\big(U(x+h) – U(x–k)\big) + \big(Q(x+h) – Q(x–k)\big)\cdot U(x–k)$" .

The complicated second-order divided-difference discretization exhibited above does not so much resolve the dilemma presented at the beginning of this appendix as it relieves us of the necessity to employ the crude first-order divided-difference discretization of §4 when gaps $h_j$ vary. The dilemma persists because, when all gaps have the same tiny width $h$ , more accurate fourth-order tridiagonal discretizations can be constructed from at most second-order finite differences. These are a story for another day.