

A Floating-Point Trick to Solve Boundary-Value Problems Faster

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

§0. Abstract: These notes resuscitate an old trick to accelerate the numerical solution of certain discretized boundary-value problems. Without the trick, half the digits carried by the arithmetic can be lost to roundoff when the discretization's grid-gaps get very small. The trick can obtain adequate accuracy from arithmetic with `float` variables 4-bytes wide instead of `double` variables 8-bytes wide. Wider data moves slower through the computer's memory system and pipelines. The trick is tricky for programs written in `MATLAB™ 7`, `JAVA`, `FORTRAN` and post-1985 `ANSI C`. The trick is easy for the original Kernighan-Ritchie `C` of the late 1970s, and for a few implementations of `C99` that fully support IEEE Standard 754 for Binary Floating-Point.

Contents:

| | | | |
|---|--|------|----|
| §1. Introduction: | $\text{div}(\mathbf{p}\cdot\text{grad } u) + q\cdot u = r$ | page | 1 |
| §2. Discretizing the Differential Equation: | Divided Differences | | 2 |
| §3. How Roundoff Corrupts the Discretization: | | | 3 |
| §4. Accurate Residuals: | | | 5 |
| §5. A Trickier Trick: | | | 6 |
| §6. Example: | $(x\cdot u')' + 4x\cdot(1-x^2)\cdot u = 0$ | | 7 |
| §7. The 2nd Program: | preserves symmetry | | 11 |
| §8. Iterative Refinement: | recovers accuracy if done right | | 13 |
| §9. Conclusions: | Hardly any programmers will ever know the tricks. | | 16 |
| §10. Appendix 1: | accuracy despite cancellation | | 17 |
| §11. Appendix 2: | tridiagonal and 2nd order despite variable gaps | | 19 |

§1. Introduction: The solution $u(\mathbf{x})$ of a differential equation $\text{div}(\mathbf{p}\cdot\text{grad } u) + q\cdot u = r$ is often a potential computed only to permit the subsequent computation of a vector force-field $\text{grad } u$ 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 $\text{grad } u$ as accurately as it is needed. This accuracy is achieved by computing u over a sufficiently refined grid of mesh-points. As we shall see, mesh refinement can exacerbate the contamination of u by roundoff unless the program acts to abate that contamination. By far the simplest abatement resorts to extra-precise arithmetic. When extra precision is unavailable or too slow, the abatement must use a tricky trick exhibited in these notes. An example will show how well it works to compute the regular solution of a singular differential equation.

The trick will be 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's application to elliptic *partial* differential equations, or to higher-order differential equations that determine an equilibrium solution U , entails routine elaboration needing no explanation. Further elaborating the trick to apply to parabolic and hyperbolic partial differential equations that characterize propagation may incur enough additional memory traffic to vitiate the trick; but that's a story for another day.

§2. Discretizing the Differential Equation: 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 $x_0 < x_1 < x_2 < \dots < x_N$ 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, $p_{j+1/2} := P((x_j + x_{j+1})/2)$, $\mathbf{q}_j := Q(x_j, \mathbf{u}_j)$ 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

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

(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 $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ of equations in which \mathbf{u} is a column of unknowns \mathbf{u}_j , $\text{Diag}(\mathbf{q})$ 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 T is a tridiagonal matrix computed from the elements $p_{j+1/2}$ and gaps h_j . The bottom and topmost entries in $T + \text{Diag}(\mathbf{q})$ and \mathbf{r} include contributions from the boundary-value problem's boundary conditions.

Sometimes $\text{Diag}(\mathbf{q})$ is supplanted by a tridiagonal matrix to help approximate the differential equation better. For the same reason 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 $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ has to be solved for the desired $\mathbf{u} = (T + \text{Diag}(\mathbf{q}))^{-1} \cdot \mathbf{r}$. 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 $T + \text{Diag}(\mathbf{q}) \approx E \cdot B$ wherein B is bidiagonal and upper-triangular, and 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 $\mathbf{u} \approx B^{-1} \cdot (E^{-1} \cdot \mathbf{r})$ by first forward substitution (perhaps permuted) to compute $E^{-1} \cdot \mathbf{r}$ 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 $\Delta \mathbf{u} \approx B^{-1} \cdot (E^{-1} \cdot (\mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}))$ using the latest values for \mathbf{u} , \mathbf{q} and \mathbf{r} but the same factors E and B as before.
- If \mathbf{q} and \mathbf{r} depend weakly but not very upon \mathbf{u} then $\Delta \mathbf{u} \approx B^{-1} \cdot (E^{-1} \cdot (\mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}))$ using the latest values \mathbf{u} , \mathbf{q} and \mathbf{r} and a recomputed factorization $T + \text{Diag}(\mathbf{q}) \approx E \cdot B$.
- If \mathbf{q} and \mathbf{r} depend strongly upon \mathbf{u} then $\Delta \mathbf{u} \approx B^{-1} \cdot (E^{-1} \cdot (\mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}))$ using the latest values for \mathbf{u} , \mathbf{q} and \mathbf{r} and recomputed factors E and 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 $\mathbf{u} \rightarrow \mathbf{u} + \Delta\mathbf{u}$ should suffice to solve the equation $(\mathbf{T} + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ for \mathbf{u} as accurately as the data \mathbf{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 $\mathbf{s} := \mathbf{r} - (\mathbf{T} + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$. 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 “ $\mathbf{r} - (\mathbf{T} + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ ” 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 $\mathbf{T} = \mathbf{T}^T$ symmetrical. Sometimes P and consequently \mathbf{T} depend upon \mathbf{u} , contrary to our assumptions. None of these possibilities affect the trick.

Finally, the j^{th} row $(\mathbf{T} \cdot \mathbf{u})_j$ of $\mathbf{T} \cdot \mathbf{u}$ always has the form $a_{j-1} \cdot \mathbf{u}_{j-1} - b_j \cdot \mathbf{u}_j + c_{j+1} \cdot \mathbf{u}_{j+1}$ 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 $|\mathbf{q}_j| / (|a_{j-1}| + |b_j| + |c_{j+1}|) \rightarrow 0$ and $|\mathbf{r}_j| / (|a_{j-1}| + |b_j| + |c_{j+1}|) \rightarrow 0$
roughly like $(h_{j-1} + h_j) \cdot \max\{h_{j-1}, h_j\} \rightarrow 0$.
- 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, $N \rightarrow \infty$ when $\max\{h_{j-1}, h_j\} \rightarrow 0$, 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 $p_{j \pm 1/2}$ of the smooth function P ; but occasional exceptions may violate property 2). None of these possibilities affect the trick.

§3. How Roundoff Corrupts the Discretization: It contributes uncertainty to almost every step of the solution process. Let ϵ denote the roundoff threshold for rational floating-point arithmetic operations. When every operation rounds to the 24 sig. bits of `float`, $\epsilon = 1/2^{24}$; ... to 53 sig. bit `double`, $\epsilon = 1/2^{53}$. Then rounded values computed from expressions like “ $v \cdot w$ ”, “ v/w ”, “ $v+w$ ” and “ $v-w$ ” lie in the respective ranges $(v \cdot w) \cdot (1 \pm \epsilon)$, $(v/w) \cdot (1 \pm \epsilon)$, $(v+w)/(1 \pm \epsilon)$ and $(v-w)/(1 \pm \epsilon)$. The smaller is ϵ , 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 $\mathbf{T} + \text{Diag}(\mathbf{q})$, and then more of them turn its factorization into $\mathbf{E} \cdot \mathbf{B} = \mathbf{T} + \text{Diag}(\mathbf{q}) \pm \epsilon \cdot (|\mathbf{E}| \cdot |\mathbf{B}| + |\mathbf{T}| + \text{Diag}(|\mathbf{q}|))$ roughly; the uncertainty here is dominated by the contributions from $\pm \epsilon \cdot (|\mathbf{E}| \cdot |\mathbf{B}| + |\mathbf{T}|)$, 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 $\mathbf{T} + \text{Diag}(\mathbf{q})$ 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 §8. 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 $(T + \text{Diag}(\mathbf{q}) \pm \varepsilon \cdot (|E| \cdot |B| + |T| + \text{Diag}(|\mathbf{q}|))) \cdot \mathbf{u} = \mathbf{r}$, not $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ as desired. We can ignore the difference if $T + \text{Diag}(\mathbf{q})$ is far enough from singular and if ε 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 the residual $\mathbf{s} := \mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ 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 $\mathbf{s} := \mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ 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 $\text{Diag}(\mathbf{q}_j)$ in the equation $T \cdot \mathbf{u} + \text{Diag}(\mathbf{q}) \cdot \mathbf{u} = \mathbf{r}$ that has to be solved for \mathbf{u} , the j^{th} element $(T \cdot \mathbf{u})_j := a_{j-1} \cdot \mathbf{u}_{j-1} - b_j \cdot \mathbf{u}_j + c_{j+1} \cdot \mathbf{u}_{j+1}$ 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 $(T \cdot \mathbf{u})_j$ must cancel down to something small, roughly of the order of

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

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

Coincidentally, when $(T \cdot \mathbf{u})_j$ 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 $\pm \varepsilon \cdot (|T| \cdot |\mathbf{u}|)_j$ to $(T \cdot \mathbf{u})_j$. This term dominates the computed residual's uncertainty due to roundoff: $\mathbf{s} - \mathbf{r} + (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \pm \varepsilon \cdot (|T| + \text{Diag}(|\mathbf{q}|)) \cdot |\mathbf{u}|$ roughly within a factor of 2 or 3 regardless of whether $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ is computed after $T + \text{Diag}(\mathbf{q})$ is or computed separately as $T \cdot \mathbf{u} + \text{Diag}(\mathbf{q}) \cdot \mathbf{u}$. Even if the computed residual vanishes the current estimate \mathbf{u} must satisfy $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r} \pm \varepsilon \cdot (|T| + \text{Diag}(|\mathbf{q}|)) \cdot |\mathbf{u}|$, an equation perturbed by uncertainty dominated by the term $\pm \varepsilon \cdot |T| \cdot |\mathbf{u}|$. This is almost as bad as the uncertainty $\pm \varepsilon \cdot (|E| \cdot |B| + |T| + \text{Diag}(|\mathbf{q}|)) \cdot |\mathbf{u}|$ that afflicted Gaussian elimination and triangular factorization unless an unfortunate choice of pivots bloated the triangular factors E and B . This dominant term $|T| \cdot |\mathbf{u}|$ is bigger than all the other terms \mathbf{r} , $\text{Diag}(|\mathbf{q}|) \cdot |\mathbf{u}|$ and even $|T \cdot \mathbf{u}|$ by factors like $1 / ((h_{j-1} + h_j) \cdot \max\{h_{j-1}, h_j\})$ 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 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 fraction of extravagantly too many digits will be accurate enough.

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 \mathbf{u} about as accurately as the given data determine it. The trick is to compute residuals \mathbf{s} well enough.

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

For definiteness let us assume all the given data, namely $\{x_j\}$ and columns \mathbf{q} and \mathbf{r} and arrays $\{a_j\}$ and $\{c_j\}$ of off-diagonal elements of \mathbf{T} , and the columns \mathbf{u} of a putative solution and its residual \mathbf{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 \mathbf{u} and $\Delta \mathbf{u}$ then we store also the two arrays of subdiagonal elements of \mathbf{E} and diagonal elements of \mathbf{B} as `floats` and write `float` $\Delta \mathbf{u}$ over \mathbf{s} .

The array $\mathbf{g} := \text{diag}(\mathbf{T} + \text{Diag}(\mathbf{q}))$ can be treated in any of several ways: One is to compute each element $g_j := q_j - a_{j-1} - c_{j+1}$ 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 store the array \mathbf{g} as an array of 8-byte `doubles`. Trickier ways to cope with \mathbf{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 \mathbf{C} and is now available from some implementations of $\mathbf{C99}$, every arithmetic operation is rounded to at least `double` regardless of whether its operands are `floats` or `doubles`. In this environment the `float` residual \mathbf{s} can be computed amply accurately from any one of the three assignments

$$\begin{aligned} & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot \mathbf{u}_{j-1} - \mathbf{g}_j \cdot \mathbf{u}_j - c_{j+1} \cdot \mathbf{u}_{j+1} \text{" } \quad \text{or} \\ & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot \mathbf{u}_{j-1} - (\mathbf{q}_j - a_{j-1} - c_{j+1}) \cdot \mathbf{u}_j - c_{j+1} \cdot \mathbf{u}_{j+1} \text{" } \quad \text{or} \\ & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot (\mathbf{u}_{j-1} - \mathbf{u}_j) - c_{j+1} \cdot (\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j \cdot \mathbf{u}_j \text{" } \end{aligned}$$

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 \mathbf{JAVA} or $\mathbf{ANSII C}$ (1987), the foregoing assignments must be encumbered by *casts* (conversions to `double`) to achieve amply adequate accuracy thus:

$$\begin{aligned} & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot (\text{double})\mathbf{u}_{j-1} - \mathbf{g}_j \cdot (\text{double})\mathbf{u}_j - c_{j+1} \cdot (\text{double})\mathbf{u}_{j+1} \text{" } \quad \text{or} \\ & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot (\text{double})\mathbf{u}_{j-1} - (((\text{double})\mathbf{q}_j - a_{j-1}) - a_j) \cdot \mathbf{u}_j - c_{j+1} \cdot (\text{double})\mathbf{u}_{j+1} \text{" } \quad \text{or} \\ & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot ((\text{double})\mathbf{u}_{j-1} - \mathbf{u}_j) - c_{j+1} \cdot ((\text{double})\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j \cdot (\text{double})\mathbf{u}_j \text{" } . \end{aligned}$$

(DON'T DISREGARD PARENTHESES!)

§5. 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:

$$"s_j := r_j - a_{j-1} \cdot ((u_{j+1} - u_j) - (u_j - u_{j-1})) - (c_{j+1} - a_{j-1}) \cdot (u_{j+1} - u_j) - q_j \cdot u_j"$$

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 u_{j-1}/u_j to stay close to 1, and to come closer as all the gaps h_j shrink. Then each subtraction $c_{j+1} - a_{j-1}$, $u_{j+1} - u_j$, and usually $(u_{j+1} - u_j) - (u_j - u_{j-1})$ 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 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 §2, namely

$$2 \cdot (p_{j+1/2} \cdot (u_{j+1} - u_j) / h_j - p_{j-1/2} \cdot (u_j - u_{j-1}) / h_{j-1}) / (h_j + h_{j-1}) \approx (P \cdot U)'$$

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 ϵ and the gaps h_j all of order h , say. Then $(u_{j+1} - u_j) / h_j$ is of order 1 because it approximates $U'(x_j)$ roughly; the quotient introduces no *new* roundoff. The multiplication in $p_{j+1/2} \cdot (u_{j+1} - u_j) / h_j$ suffers a new rounding error of order ϵ , 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 ϵ/h from roundoff.

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

$$2 \cdot (p_{j+1/2} \cdot ((u_{j+1} - u_j) / h_j - (u_j - u_{j-1}) / h_{j-1}) + (p_{j+1/2} - p_{j-1/2}) \cdot (u_j - u_{j-1}) / h_{j-1}) / (h_j + h_{j-1})$$

that costs two extra subtractions neither of which introduces a new rounding error. The first extra subtraction $(u_{j+1} - u_j) / h_j - (u_j - u_{j-1}) / h_{j-1} \approx U'(x_j + h_j/2) - U'(x_j - h_{j-1}/2) \approx U''(x_j) \cdot (h_j + h_{j-1}) / 2$ very roughly, so it is of order h and the subsequent multiplication suffers a rounding error of order $h \cdot \epsilon$. 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 $\epsilon \cdot h$. The subsequent addition and then division by $(h_j + h_{j-1}) / 2$ make the new formula's total contribution from roundoff of order ϵ , not ϵ/h .

Thus are computed residuals $\mathbf{s} := \mathbf{r} - (\mathbf{T} + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ contaminated by tiny practically irreducible rounding errors of the order of $\varepsilon \cdot (|\mathbf{T} \cdot \mathbf{u}| + \text{Diag}(|\mathbf{q}|) \cdot |\mathbf{u}| + |\mathbf{r}|) \approx O(\varepsilon)$ instead of huge rounding errors of the order of $\varepsilon \cdot (|\mathbf{T}| \cdot |\mathbf{u}| + \text{Diag}(|\mathbf{q}|) \cdot |\mathbf{u}| + |\mathbf{r}|) \approx O(\varepsilon/h^2)$ 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_{\dots} 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 $\text{div}(\mathbf{p} \cdot \mathbf{grad} \, u) + q \cdot u = r$. 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 are 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.

§6. 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 $u(\pm 1) = 1$ 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

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

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 $v(\pm 1) = 0$ 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 $N \gg 2$ and set $x_j := j/N - 1$ for $j := 0, 1, 2, \dots, 2N-1, 2N$. 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 $\mathbf{T} + \text{Diag}(\mathbf{q})$ with $[g_1, g_2, \dots, g_{2N-1}]$ on its main diagonal and $[c_2, c_3, \dots, c_{2N-1}] = [a_1, a_2, \dots, a_{2N-2}]$ 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 $r_{2N-1} = -c_{2N} \cdot \mathbf{u}_{2N} = -a_{2N-1} = a_0$ that convey the boundary conditions $u(\pm 1) := 1$.

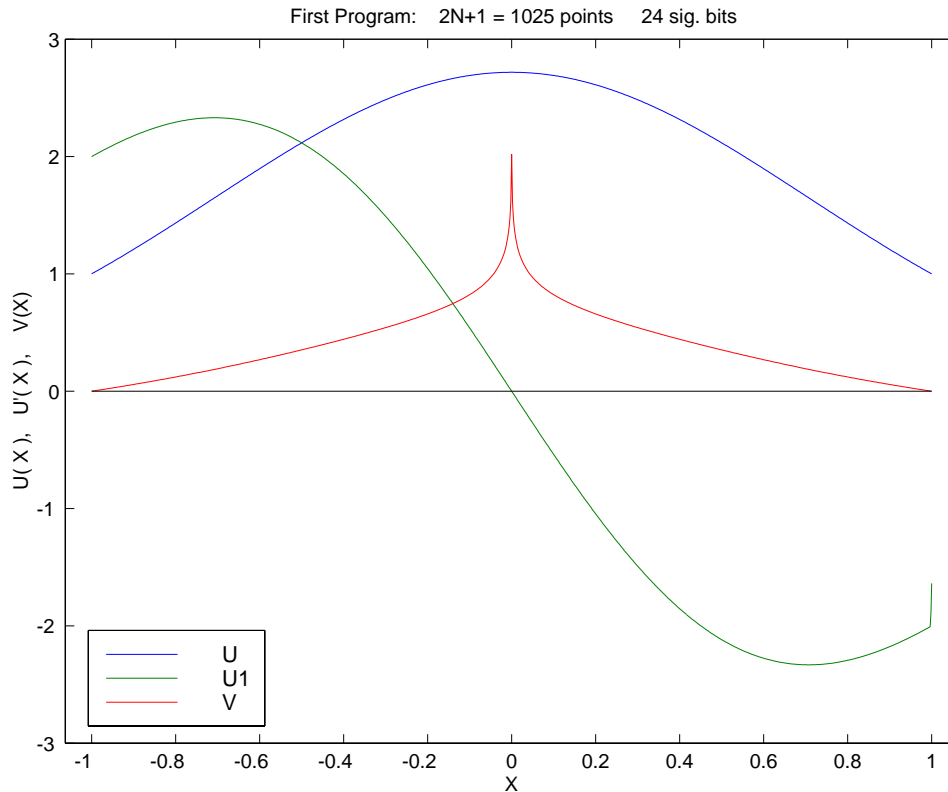
Elements $[\mathbf{u}_1; \mathbf{u}_2; \dots; \mathbf{u}_{2N-1}]$ of column \mathbf{u} should be computed by solving $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$. The first attempt factorizes $T + \text{Diag}(\mathbf{q}) = E \cdot B$ wherein B is bidiagonal with $[\beta_1, \beta_2, \dots, \beta_{2N-1}]$ on its diagonal and $[a_1, a_2, \dots, a_{2N-2}]$ on its first superdiagonal, and E is bidiagonal with 1 everywhere on its diagonal and $[e_1, e_2, \dots, e_{2N-2}]$ on its first subdiagonal. A recurrence produces $e_{j-1} := a_{j-1}/\beta_{j-1}$ and $\beta_j := g_j - a_{j-1} \cdot e_{j-1}$ for $j = 2, 3, \dots, 2N-1$ in turn starting with $\beta_1 := g_1$. This amounts to Gaussian elimination without pivotal exchanges. Its simplicity of programming could be offset by the appearance of a tiny β_{j-1} followed by a huge β_j that greatly amplifies roundoff. Numerical degradation like that has yet to occur during the foregoing computation. However the last $\beta_{2N-1} = 0$ except for roundoff because $T + \text{Diag}(\mathbf{q})$ is singular (not invertible), which complicates the first attempt to compute \mathbf{u} .

To see why $T + \text{Diag}(\mathbf{q})$ is singular let $\Xi = \Xi^T$ be the matrix obtained from the $2N-1$ -by- $2N-1$ identity by reversing the order of its rows (or columns) and observe that $\Xi \cdot (T + \text{Diag}(\mathbf{q})) \cdot \Xi = -(T + \text{Diag}(\mathbf{q}))$, whereupon $\det(\Xi)^2 \cdot \det(T + \text{Diag}(\mathbf{q})) = (-1)^{2N-1} \cdot \det(T + \text{Diag}(\mathbf{q})) = 0$. Then $\det(B) = \det(E) \cdot \det(B) = \det(T + \text{Diag}(\mathbf{q})) = 0$ too, which explains why $\beta_{2N-1} = 0$. Despite that $T + \text{Diag}(\mathbf{q})$ is singular, the equation $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ 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 $\mathbf{v} = \Xi \cdot \mathbf{v} \neq \mathbf{0}$ of $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{v} = \mathbf{0}$ with a sharp spike in the middle of it.

Any first attempt to solve $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ naively gets \mathbf{u} contaminated by the addition of some arbitrary or infinite multiple of the spiked singular solution \mathbf{v} of $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{v} = \mathbf{0} \neq \mathbf{v}$. If that multiple is small enough, the spike is narrow enough to go unnoticed until too late. How can the smooth regular solution $u(x)$ be separated numerically from spiked singular solutions $v(x)$ that satisfy the same boundary conditions and even the symmetry condition $v(-x) \equiv v(x)$?

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 $u'(x)/x \rightarrow u''(0)$ as $x \rightarrow 0$; and then substitution into the differential equation implies $u''(0) = -2u(0)$. This *internal boundary condition* upon $u(x)$ 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 $(T + \text{Diag}(\mathbf{q})) \cdot \hat{\mathbf{u}} = \mathbf{r}$, can be imposed upon a revised solution $\mathbf{u} := \hat{\mathbf{u}} - \lambda \cdot \mathbf{v}$ by choosing λ aptly after computing a singular solution $\mathbf{v} \neq \mathbf{0}$ of $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{v} = \mathbf{0}$. Our first program does this.

1st Program's Details: The computed solution of $B \cdot \hat{\mathbf{u}} = E^{-1} \cdot \mathbf{r}$ is $\hat{\mathbf{u}}$; its last component is set to $\hat{u}_{2N-1} := 1 + 2/N$ to filter out most of the singular solution \mathbf{v} . This is the solution of $B \cdot \mathbf{v} = \mathbf{0}$; its last component $v_{2N-1} := 1/(2N-1)$ is set to keep the spike v_N roughly between 1 and 4. Then λ 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 $u'(x_j)$ is approximated within $O(1/N^2)$ by $\mathbf{u}_1_j := (\mathbf{u}_{j+1} - \mathbf{u}_{j-1}) \cdot N/2$ except at the boundaries where $u'(-1) \approx \mathbf{u}_1_0 := (4\mathbf{u}_1 - 3\mathbf{u}_0 - \mathbf{u}_2) \cdot N/2$. The errors $\max_j |\mathbf{u}_j - u(x_j)|$ in \mathbf{u} and $\max_j |\mathbf{u}_1_j - u'(x_j)|$ in \mathbf{u}_1 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 $\mathbf{u} \approx u(x)$, $\mathbf{u1} \approx u'(x)$ and $\mathbf{v} \approx v(x)$ carrying 24 sig. bitsResults from the First Program carrying 24 sig. bits ($\epsilon \approx 6/10^8$)

| N | err(\mathbf{u}) | err(\mathbf{u}) $\cdot N^2$ | err($\mathbf{u1}$) | err($\mathbf{u1}$) $\cdot N^2$ |
|-----|---------------------|---------------------------------|----------------------|----------------------------------|
| 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 |

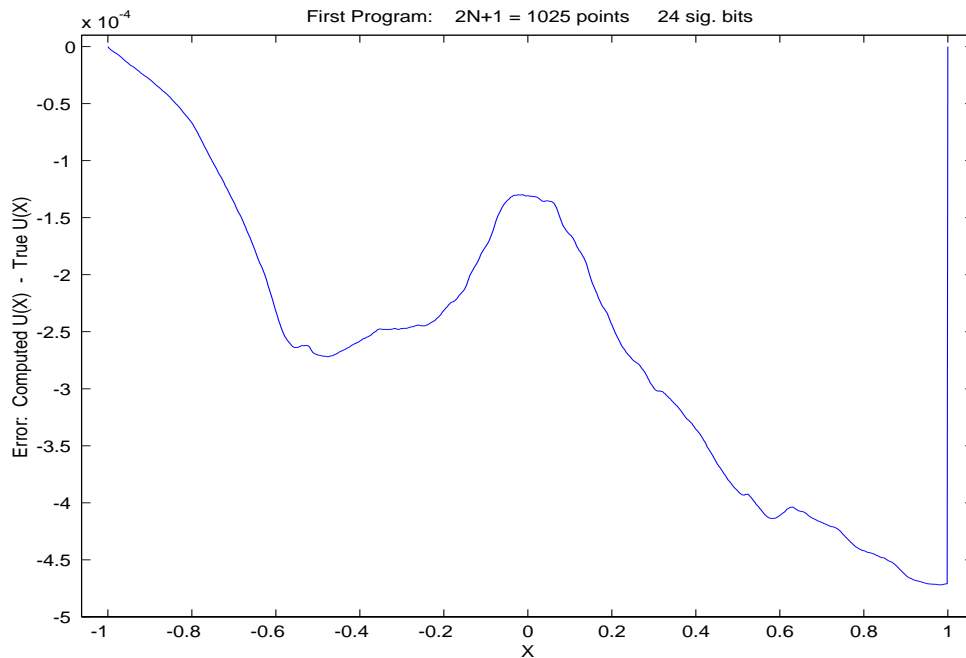
$$N = \text{\#gaps}, \quad \text{err}(\mathbf{u}) := \max_j |\mathbf{u}_j - u(x_j)|, \quad \text{err}(\mathbf{u1}) := \max_j |\mathbf{u1}_j - u'(x_j)|.$$

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

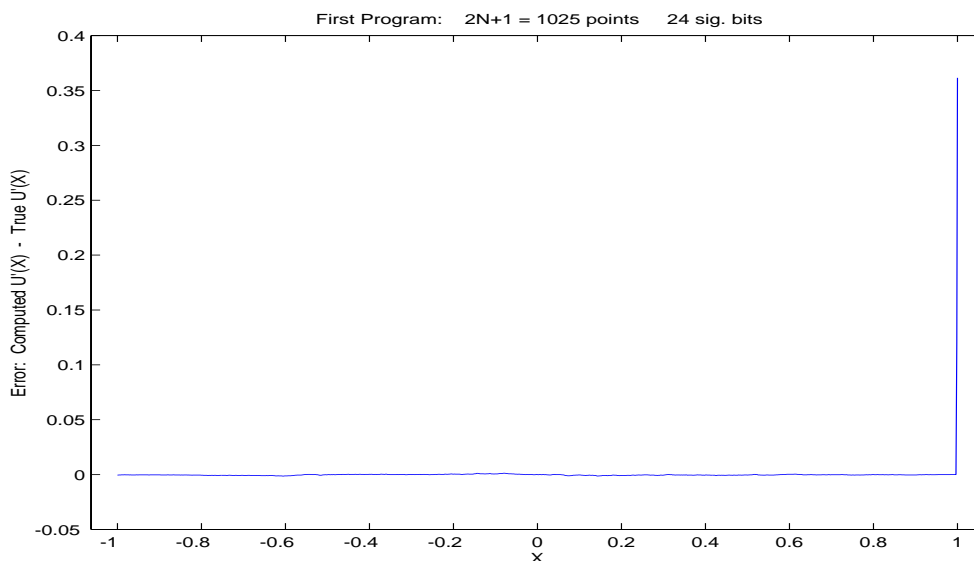
\mathbf{u} never gets much below about $\sqrt{\epsilon}$, and making N too big actually worsens the total error by amplifying roundoff. Usually. Roundoff contributes raggedly (*not randomly*) to the total error.

The gradient's estimate \mathbf{u}_1 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 $\mathbf{u}_{1_{2N}} - u'(1)$ overwhelmingly bigger than every other error $\mathbf{u}_{1_j} - u'(x_j)$ for $0 \leq j < 2N$.

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



Error in the First Program's $\mathbf{u}_1 \approx u'(x)$ 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 $\mathbf{u1}$ 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 $\mathbf{u1}$'s error.

§7. 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 $-1 < x \leq 0$, whence $\mathbf{u}_j \approx u(x_j)$ need be computed only for $1 \leq j \leq N$.

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 $u(x) = \sum_{n \geq 0} \mu_n \cdot x^n / n!$ starting with an arbitrary $\mu_0 = u(0) \neq 0$ though $\mu_1 = u'(0) = 0$; the other coefficients μ_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 $w(x) \neq 0$ 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 $0 < x < \zeta \leq 1$. And then we would find that

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

Consequently $w(x) = 0$ at least for $0 \leq x \leq 1$; beyond $x = 1$ the differential equation is regular and determines its solution $w(x) = 0$ uniquely for all $x \geq 0$ 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 $u(\pm 1) = 1$ 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 to the differential equation's discretization defines its solution uniquely; so its new matrix $T + \text{Diag}(\mathbf{q})$ 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 $j := 0, 1, 2, \dots, N-1, N$. 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 $1 \leq j \leq N$, and $g_j := \mathbf{q}_j - a_{j-1} - c_{j+1}$ again but now for $1 \leq j \leq N-1$. 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 $c_{N+1} \cdot \mathbf{u}_{N+1}$. The external boundary condition is $\mathbf{u}_0 := 1$.

Our second attempt to compute the elements $[\mathbf{u}_1; \mathbf{u}_2; \dots; \mathbf{u}_N]$ of column \mathbf{u} constructs a new symmetric tridiagonal matrix $T + \text{Diag}(\mathbf{q})$ with $[g_1, g_2, \dots, g_{N-1}, g_N]$ on its main diagonal and $[c_2, c_3, \dots, c_N] = [a_1, a_2, \dots, a_{N-1}]$ 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 $(T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u} = \mathbf{r}$ we first factorize $T + \text{Diag}(\mathbf{q}) = E \cdot B$ wherein B is bidiagonal with $[\beta_1, \beta_2, \dots, \beta_N]$ on its diagonal and $[a_1, a_2, \dots, a_{N-1}]$ on its first superdiagonal, and E is bidiagonal with 1 everywhere on its diagonal and $[e_1, e_2, \dots, e_{N-1}]$ with $e_j := a_j / \beta_j$ on its first subdiagonal. The numbers β_j are computed from a recurrence $\beta_j := g_j - a_{j-1} \cdot e_{j-1}$ for $j = 2, 3, \dots, N$ in turn starting with $\beta_1 := g_1$. This amounts to the same Gaussian elimination without pivotal exchanges as before; however, since the new $T + \text{Diag}(\mathbf{q})$ turns out to be positive definite there is no risk now that some tiny pivot β_{j-1} will be followed by an enormous β_j .

Forward substitution computes $\mathbf{w} := E^{-1} \cdot \mathbf{r}$: $w_1 := r_1$ and $w_j := -e_{j-1} \cdot w_{j-1}$. Subsequent back-substitution computes $\mathbf{u} := B^{-1} \cdot \mathbf{w}$: $u_N := w_N / \beta_N$ and $u_j := (w_j - a_j \cdot u_{j+1}) / \beta_j$. Finally, gradient $u'(x)$ is approximated by $\mathbf{u1}$ as before.

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

| N | err(\mathbf{u}) | err(\mathbf{u}) $\cdot N^2$ | err($\mathbf{u1}$) | err($\mathbf{u1}$) $\cdot N^2$ |
|-----|---------------------|---------------------------------|----------------------|----------------------------------|
| 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 |

$N = \# \text{gaps}$, $\text{err}(\mathbf{u}) := \max_j |\mathbf{u}_j - u(x_j)|$, $\text{err}(\mathbf{u1}) := \max_j |\mathbf{u1}_j - u'(x_j)|$.

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 $\sqrt{\epsilon}$. Less roundoff occurs when N is a power of 2 because then divisions by N are exact. The second program computes the approximate gradient $\mathbf{u1}$ 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}_1 , at least roughly $3.7/N^2 + 2N^3/10^{11}$, seems excessive. (The *uncertainty* is not the *error* but instead how big there is good reason to believe 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.

§8. 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 $\mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ to reveal the extent to which \mathbf{u} dissatisfies the equation we wish to solve. The third program assigns " $\mathbf{s} := \mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ " 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 " $\mathbf{r} - (T + \text{Diag}(\mathbf{q})) \cdot \mathbf{u}$ " and the triangular factorization $T + \text{Diag}(\mathbf{q}) = E \cdot B$ 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}_1 , \mathbf{q} , \mathbf{r} , $\{a_j\}$, $\{\beta_j\}$ 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 from some implementations of *C99*, the float residual \mathbf{s} can be computed amply accurately from any one of the three assignments

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

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:

$$\begin{aligned} & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot (\text{double})\mathbf{u}_{j-1} - g_j \cdot \mathbf{u}_j - a_j \cdot (\text{double})\mathbf{u}_{j+1} \text{" or} \\ & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot (\text{double})\mathbf{u}_{j-1} - (((\text{double})\mathbf{q}_j - a_{j-1}) - a_j) \cdot \mathbf{u}_j - a_j \cdot (\text{double})\mathbf{u}_{j+1} \text{" or} \\ & \text{" } \mathbf{s}_j := \mathbf{r}_j - a_{j-1} \cdot ((\text{double})\mathbf{u}_{j-1} - \mathbf{u}_j) - a_j \cdot ((\text{double})\mathbf{u}_{j+1} - \mathbf{u}_j) - \mathbf{q}_j \cdot (\text{double})\mathbf{u}_j \text{"}. \end{aligned}$$

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:

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

DO NOT DISREGARD 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 `FORTRAN`nish environments, using `double` arithmetic upon `float` variables; those formulas produce slightly better results (not tabulated below) with no need for the analysis in §5 nor Appendix 1.

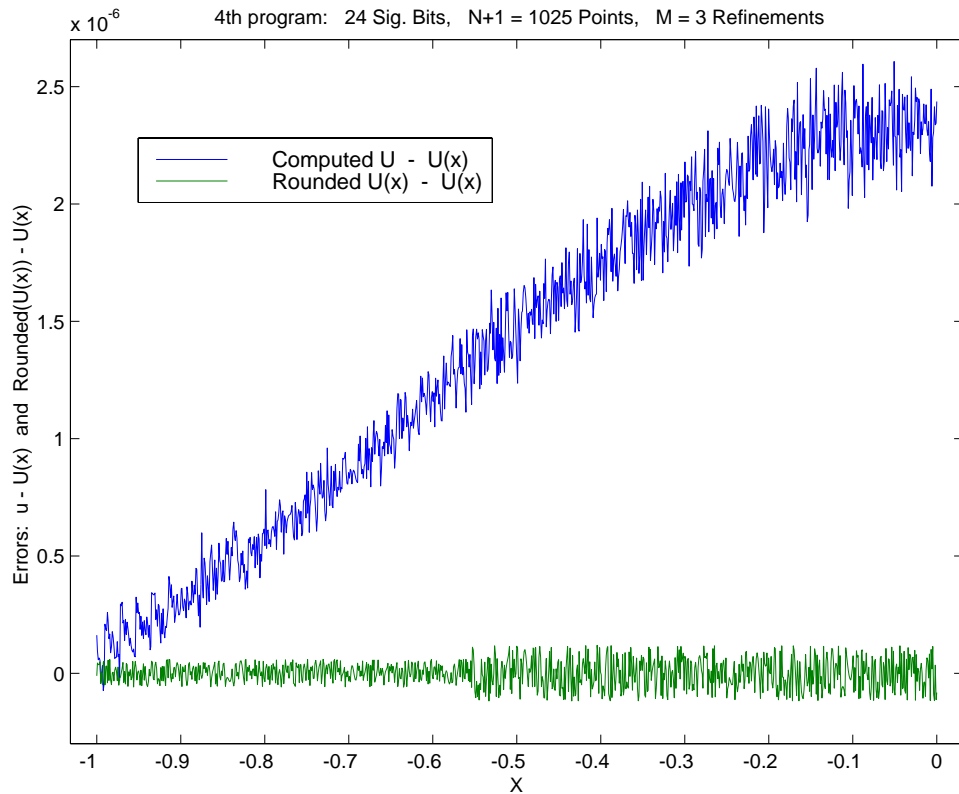
Results from the Fourth Program carrying 24 sig. bits ($\epsilon \approx 6/10^8$)

| N | M | err(u) | err(u)·N ² | err(u1) | err(u1)·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 | 48 |
| | 1 & 2 | 0.0010349 | 2.38 | 0.001612 | 3.71 | 1 & 2 | |
| 64 | 0 | 0.000663 | 2.73 | 0.00099 | 4.0 | 0 | 64 |
| | 1 & 2 | 0.0005821 | 2.38 | 0.000904 | 3.70 | 1 & 2 | |
| 96 | 0 | 0.000118 | 1.09 | 0.00022 | 2.0 | 0 | 96 |
| | 1 & 2 | 0.0002586 | 2.38 | 0.000393 | 3.62 | 1 & 2 | |
| 128 | 0 | 0.000073 | 1.19 | 0.00027 | 4.5 | 0 | 128 |
| | 1 & 2 | 0.0001456 | 2.39 | 0.000206 | 3.38 | 1 & 2 | |
| 192 | 0 | 0.000531 | 19.56 | 0.00102 | 37.7 | 0 | 192 |
| | 1 & 2 | 0.0000646 | 2.38 | 0.000107 | 3.94 | 1 & 2 | |
| 256 | 0 | 0.000095 | 6.24 | 0.00037 | 24.4 | 0 | 256 |
| | 1 & 2 | 0.0000364 | 2.39 | 0.000061 | 4.00 | 1 & 2 | |
| 384 | 0 | 0.000394 | 58.09 | 0.00107 | 157.3 | 0 | 384 |
| | 1 | 0.0000162 | 2.39 | 0.000049 | 7.27 | 1 | |
| | 2 & 3 | 0.0000162 | 2.39 | 0.000053 | 7.81 | 2 & 3 | |
| 512 | 0 | 0.000338 | 88.59 | 0.00202 | 528.3 | 0 | 512 |
| | 1 | 0.0000091 | 2.38 | 0.000061 | 16.11 | 1 | |
| | 2 & 3 | 0.0000092 | 2.41 | 0.000065 | 16.92 | 2 & 3 | |
| 768 | 0 | 0.006888 | 4062.71 | 0.01578 | 9310.2 | 0 | 768 |
| | 1 | 0.0000156 | 9.20 | 0.000089 | 52.50 | 1 | |
| | 2 & 3 | 0.0000041 | 2.41 | 0.000088 | 51.75 | 2 & 3 | |
| 1024 | 0 | 0.0037995 | 3984.06 | 0.012934 | 13561.9 | 0 | 1024 |
| | 1 | 0.0000062 | 6.45 | 0.000119 | 124.27 | 1 | |
| | 2 & 3 | 0.0000024 | 2.49 | 0.000114 | 119.36 | 2 & 3 | |
| 1536 | 0 | 0.0091051 | 21481.52 | 0.021961 | 531.6 | 0 | 1536 |
| | 1 | 0.0000313 | 73.85 | 0.000225 | 531.61 | 1 | |
| | 2 & 3 | 0.0000012 | 2.90 | 0.000174 | 411.38 | 2 & 3 | |
| 2048 | 0 | 0.0381667 | 160082.62 | 0.076282 | 319949.1 | 0 | 2048 |
| | 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 | |

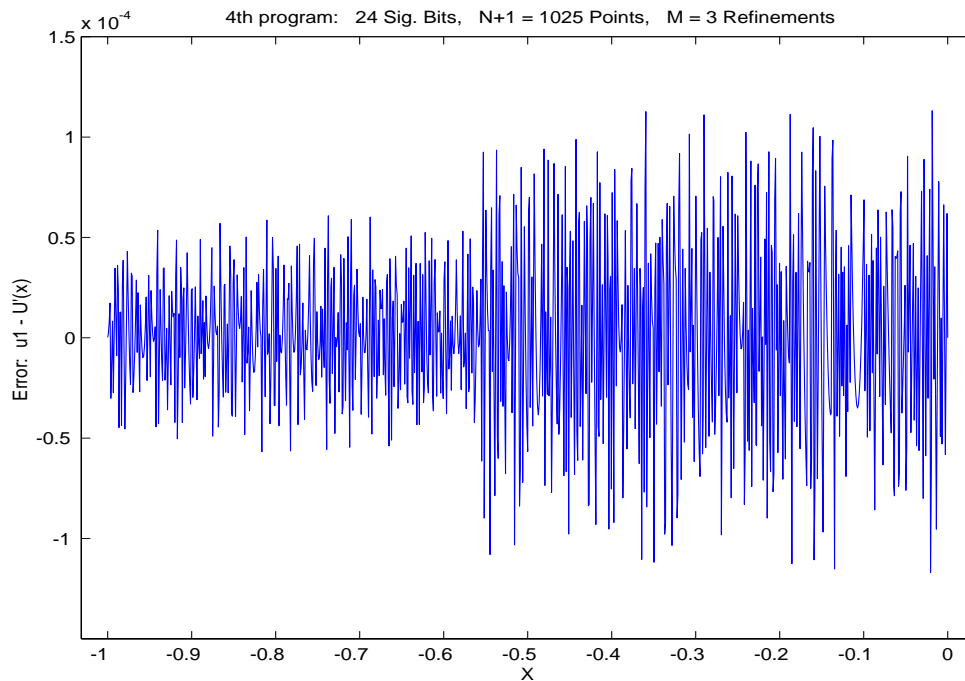
N = #gaps, M = #refinements, $\text{err}(\mathbf{u}) := \max_j |\mathbf{u}_j - u(x_j)|$, $\text{err}(\mathbf{u1}) := \max_j |\mathbf{u1}_j - u'(x_j)|$.

The dwindling accuracy of the estimated gradient $\mathbf{u1}$ 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 $\mathbf{u1} \approx u'(x)$ computed carrying 24 sig. bits, $N = 1024$, $M = 3$



The dwindling accuracy of the estimated gradient \mathbf{u}_1 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 ε 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}_1 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 tabulated results corroborate error-analyses that predict \mathbf{u}_1 '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}_1 , 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 $\kappa(\mathbf{T} + \text{Diag}(\mathbf{q})) := \|\mathbf{T} + \text{Diag}(\mathbf{q})\| \cdot \|(\mathbf{T} + \text{Diag}(\mathbf{q}))^{-1}\|$ turns out to be roughly $4N^2$ for any plausible norm $\|\dots\|$. Iterative refinement converges quickly only if roundoff disturbs the triangular factorization of $\mathbf{T} + \text{Diag}(\mathbf{q})$ by rather less than its distance from the nearest singular matrix, and this happens only if $\varepsilon \cdot \kappa(\mathbf{T} + \text{Diag}(\mathbf{q})) \ll 1$. This implies that convergence is too likely to go slow unless $N \ll 1/\sqrt{4\varepsilon} = 2048$. The bound is corroborated by the tabulated results, which show 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. But no such methods work upon the partial differential equations for which this note's trick is intended. Another method that works well upon our example is *Collocation of Splines*; this method is implemented in recent MATLABS' `bvp4c`. Analogous methods for partial differential equations are too complicated to get used much.

§9. Conclusions: These notes' thesis, supported by analysis and an example, is that boundary-value problems $\text{div}(\mathbf{p} \cdot \text{grad } u) + q \cdot u = r$ 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, which is what current graphics processors do. Residuals must be computed accurately enough via a trick effortless only if cancelling residuals are computed with 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 example was chosen to illustrate three additional phenomena:

- Without tricks, `floats` are not accurate enough 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 excessively high density incurs retardation of iterative refinement's convergence.

The trick presented in these notes is a palliative, not a cure 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, widespread 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.

§10. Appendix 1: This concerns a trick to compute scalar $z := A \cdot x - B \cdot y$ more accurately when it cancels severely enough that $|z| \ll Z := |A \cdot x| + |B \cdot y|$ 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 ε is the roundoff threshold. But whenever both $1/2 \leq e^{-2\theta} \leq A/B \leq e^{2\theta} \leq 2$ and $1/2 \leq e^{-2\phi} \leq x/y \leq e^{2\phi} \leq 2$, 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 θ and ϕ are both tiny of order h_{\dots} , 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 $A = a \cdot (1 \pm \epsilon)$ and $B = b \cdot (1 \pm \epsilon)$ 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 $z = a \cdot x - b \cdot y \pm \epsilon \cdot (|a \cdot x| + |b \cdot y|)$ would inherit uncertainty $\pm \epsilon \cdot (|a \cdot x| + |b \cdot y|)$ almost the same as the uncertainty $\pm \epsilon \cdot (|A \cdot x| + |B \cdot y|)$ 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

$$"z := (A-B) \cdot x + B \cdot (x-y)" \quad \text{and} \quad "z := A \cdot (x-y) + (A-B) \cdot y"$$
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 $1/2 \leq A/B \leq 2$ and $1/2 \leq x/y \leq 2$ 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

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

when rounding errors in the two multiplications and the addition are taken into account. Thus, to compare the trick's $z \pm \epsilon \cdot (\zeta + |z|)$ with the naively computed $z \pm \epsilon \cdot Z$ 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 $e^{-2\theta} \leq A/B \leq e^{2\theta}$ and $e^{-2\phi} \leq x/y \leq e^{2\phi}$, and θ and ϕ are both tiny.

To simplify the comparisons without loss of generality we assume first that $0 < B \leq A \leq e^{2\theta} \cdot B$; 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 $0 < y \leq x \leq e^{2\phi} \cdot y$ or else $0 < x \leq y \leq e^{2\phi} \cdot x$. In both cases we find that

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

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

Consequently, as $\theta \rightarrow 0$ and $\phi \rightarrow 0$ independently, the ratio $(\zeta + |z|)/Z$ of the trick's to the naive evaluation's uncertainties tends to zero at least as fast as $2 \cdot \tanh(\theta + \phi)$ 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, evaluate every arithmetic expression extra-precisely by default.

§11. 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 §2 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 $\mathfrak{L}U$ 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 \mathbf{u} .

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

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

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

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

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

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

$$\begin{aligned} U^\dagger(x, h) & := (U(x+h) - U(x)) / h ; & P^\dagger(x, h) & := (P(x+h) - P(x)) / h ; \\ \mathfrak{V}U(x, h) & := (P(x+h) + P(x)) \cdot U^\dagger(x, h) ; \\ \$U(x, h, k) & := (U^\dagger(x, h) - U^\dagger(x, -k)) \cdot P^\dagger(x-k, h+k) - (P^\dagger(x, h) - P^\dagger(x, -k)) \cdot U^\dagger(x-k, h+k) ; \\ \mathfrak{L}U(x, h, k) & := ((\mathfrak{V}U(x, h) - \mathfrak{V}U(x, -k)) + (1/3) \cdot (h-k) \cdot \$U(x, h, k)) / (h+k) . \end{aligned}$$

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

$$\begin{aligned} \mathfrak{L}U(x, h, k) + Q(x) \cdot U(x) + (1/3)(h-k) \cdot (Q(x+h) \cdot U(x+h) - Q(x-k) \cdot U(x-k)) / (h+k) \\ = R(x) + (1/3)(h-k) \cdot (R(x+h) - R(x-k)) / (h+k) \end{aligned}$$

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

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

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

$$(k \cdot U^\dagger(x, h) - h \cdot U^\dagger(x, k)) / (k-h) = U'(x) + O(h \cdot k).$$

If tricky suppression of roundoff in $\mathfrak{L}U(x, h, k)$ is needed, restrict gaps h_j to powers of $1/2$ as explained in §5, and change the expression “ $\mathfrak{F}U(x, h) - \mathfrak{F}U(x, -k)$ ” in the definition above of $\mathfrak{L}U(x, h, k)$ to “ $(P(x+h) - P(x-k)) \cdot U^\dagger(x, h) + (P(x-k) + P(x)) \cdot (U^\dagger(x, h) - U^\dagger(x, -k))$ ” when computing residuals. Another small improvement replaces “ $Q(x+h) \cdot U(x+h) - Q(x-k) \cdot U(x-k)$ ” by “ $Q(x+h) \cdot (U(x+h) - U(x-k)) + (Q(x+h) - Q(x-k)) \cdot U(x-k)$ ”.

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 §2 when gaps h_j vary. The dilemma persists because, when all gaps have the same tiny width h , more accurate fourth-order and yet tridiagonal discretizations can be constructed from at most second-order finite differences. These are a story for another day.