Floating-Point Tricks to Solve Boundary-Value Problems Faster

W. Kahan, Prof. Emeritus
Math. Dept., and E.E. & Computer Sci. Dept.,
University of California @ Berkeley

For UCB's Scientific & Engineering Numerical Computing Seminar
11 Sept. 2013

Abstract: Some old tricks are resuscitated 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. Tricks are tricky for programs written in MATLAB™ 7, JAVA, FORTRAN and post-1985 ANSI ***C***. For the original Kernighan-Ritchie ***C*** of the late 1970s, and for a few implementations of ***C***99 that fully support IEEE Standard 754 for Binary Floating-Point, the tricks are easy or unnecessary. Examples show how well the tricks work.

For details: `www.eecs.berkeley.edu/~wkahan/Math128/FloTrik.pdf`

Computers' memories have become HUGE
because memory has become CHEAP.

But moving data through the memory system has become COSTLY
in TIME and ENERGY DISSIPATION.

4-byte `floats` cost half as much as 8-byte `doubles` .

This motivates converting computational algorithms,
that used to be performed in `double` in past decades,
to be performed now in `float` instead..

Why not ?

Gresham's law: "*Bad* money drives out the *Good*." (from circulation)
Sir Thomas Gresham (1519 - 1579)

Gresham's law for computing:
"The *Fast* drives out the *Slow*, even if the *Fast* is wrong."

Why not supplant all `doubles` by `floats` ? *cf.* MATLAB's `eps`

Arithmetic precision of `double` : 53 sig. bits ~ 16 sig. dec. $\varepsilon \approx 2^{-52}$

of `float` : 24 sig. bits ~ 7 sig. dec. $\varepsilon \approx 2^{-23}$

7 correct sig. dec. is more than adequate accuracy
for almost all computed results used by scientists and engineers.

But what you see is not always what you get.

Roundoff corrupts the solutions of discretized differential equations, both …

• Initial-value problems: Given $T > 0$, $\mathbf{f}$ and $\mathbf{y}_0$, compute $\mathbf{y}(\tau)$ at $\tau = T$ to satisfy
$$d\mathbf{y}/d\tau = \mathbf{f}(\tau, \mathbf{y}) \text{ for } 0 \le \tau \le T \text{ and } \mathbf{y}(0) := \mathbf{y}_0 .$$

• Boundary-value problems: Given p, q, r, Φ_0 and Φ_1 , compute $\Phi(\tau)$ to satisfy
$$\mathrm{div}(p\cdot\mathbf{grad}(\Phi)) + q\cdot\Phi = r \text{ for } 0 \le \tau \le 1 \text{ and } \Phi(0) = \Phi_0 ,\ \Phi(1) = \Phi_1 .$$

and more generally when … τ runs in a 2D or 3D region … .

How does roundoff intrude into a discretization?

Discretizations:

Let θ be the step-size, or mesh-gap, of a discretization. Normally θ is very tiny.

Discretization error in computed solution $\rightarrow 0$ like θ^{Order}, depending upon … ;
Work $\rightarrow \infty$ like $1/\theta^{\text{Dimension}(\tau)\cdot\{1,\ 2 \text{ or } 3\}}$, depending upon the numerical method.
Roundoff's intrusion can grow like *Work* or faster, depending upon …" " .

Example: Initial-Value Problem $\mathbf{y}(\tau) = y_0 + \int_0^\tau \mathbf{f}(\zeta,\mathbf{y}(\zeta))\cdot d\zeta$ is approximated by …
$\mathbf{Y}(\tau+\theta) := \mathbf{Y}(\tau) + \mathbf{F}(\tau,\theta,\mathbf{Y}(\dots))\cdot\theta$ accumulated for $\tau = 0, \theta, 2\theta, 3\theta, \dots, (T/\theta)\theta$,
in which $\mathbf{F}$ estimates an average $\int_\tau^{\tau+\theta} \mathbf{f}(\zeta,\mathbf{y}(\zeta))\cdot d\zeta/\theta$ by sampling $\mathbf{f}(\dots, \mathbf{Y}(\dots))$.

Digits:

```
YYYYYYY                at   τ
+   FFFFFFF·θ             as if    fffffff
                                      lost
_______
YYYYYYY                at   τ+θ
```

A tinier step-size θ to get tinier discretization error like θ^{Order} seems to aggravate the intrusion into $\mathbf{F}$ (and hence into $\mathbf{f}$) of roundoff's uncertainty proportional to ε/θ .

This seems to limit the achievable accuracy of $\mathbf{Y}$, as if some fraction like $1/(1 + Order)$ of the arithmetic's digits of $\mathbf{f}$ were obscured by roundoff and/or discretization.

$$\mathbf{Y}(\tau+\theta) := \mathbf{Y}(\tau) + \mathbf{F}(\tau,\theta,\mathbf{Y}(\ldots))\cdot\theta \quad \text{in which} \quad \mathbf{F} \approx \int_\tau^{\tau+\theta} \mathbf{f}(\zeta,\mathbf{y}(\zeta))\cdot d\zeta/\theta + \boldsymbol{O}(\theta^{\text{Order}}) .$$

Digits:

```
YYYYYYY              at  τ
+    FFFFFFF·θ
_______
YYYYYYY              at  τ+θ
```

The lost digits FFF·θ can be retrieved by a ***Trick***: **Compensated Summation**

```
Y := y_0 ;                               … Initialization
C := o ;                                 … a column of zeros of Y's dimension
for τ = 0 to T–θ in steps of θ {
    oldY := Y ;
    ΔY := C + F(τ,θ, Y(…))·θ ;
    Y := oldY + ΔY ;                     … rounded, losing digits FFF·θ
    C := (oldY – Y) + ΔY ; }             … recovers them (DON'T REMOVE PARENTHESES)
```

Can you see why the trick works? (If $1/2 \le p/q \le 2$ then $p-q$ suffers no roundoff.)

The trick would be unnecessary if **Y** were rounded (+) and stored extra-precisely.

The trick is unnecessary also if the differential equation is so strongly stable that past errors are forgotten, or if it is so unstable that recent errors' effects are overwhelmed by the propagation of earlier errors.

Example: Over $0 \le \tau \le T$ given $T := 65/32 = 2.03125$, $v(0) := 2^{29}$, $w(0) := 0$,
solve $dv/d\tau = w/\tau$, $dw/d\tau = -4\tau\cdot(1-\tau)\cdot(1+\tau)\cdot v$ for $v(T)$.

This singular differential equation has a regular solution obtained by presubstituting 0 for 0/0 .

We shall pretend not to know that $v(T) = 2^{29}\cdot\exp(-T^2) = 8669239.890913\ldots$.

All other arithmetic is performed in 4-byte `float` (24 sig.bits).

Numerical Method: Classical 4-step 4th order *Runge-Kutta* :

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

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

The chosen number $n := 2560$ of steps produced a stepsize $\theta = T/n$ *exactly*.

Numerical Results: $V(T) = 8670448$ computed without Compensated Summation
$V(T) = 8669241$ computed with Compensated Summation
$v(T) \approx 8669240$ the true $v(T)$ rounded to 24 sig,bits.

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

Discretization of a Boundary-Value Problem

turns a second-order differential equation like

$$\mathrm{div}(p\cdot\mathbf{grad}(\Phi(\tau))) + q\cdot\Phi(\tau) = r \text{ for } \tau \text{ in region } \Omega \text{ with } \Phi \text{ specified on } \partial\Omega ,$$

into an array "$A\cdot\boldsymbol{f} = \mathbf{b}$" of difference equations for a column $\boldsymbol{f}$ whose elements f_j approximate the values of Φ at grid-points in Ω. Matrix A and column **b** depend upon grid-spacing θ and p, q, r, and the specifications of Φ on the boundary $\partial\Omega$.

We assume p, q and r vary with τ in Ω, so "$A\cdot\boldsymbol{f} = \mathbf{b}$" is linear in $\boldsymbol{f}$.
More generally, p, q and r could vary with Φ too, and then "$A\cdot\boldsymbol{f} = \mathbf{b}$" would be nonlinear in $\boldsymbol{f}$, which would complicate the exposition without changing the trick we wish to explain; therefore we assume "$A\cdot\boldsymbol{f} = \mathbf{b}$" is linear in $\boldsymbol{f}$ to keep the exposition simpler.

Here is what matters:

As the grid-spacing θ gets smaller, so does the discretization error $\boldsymbol{O}(\theta^{Order})$, but matrix A's ill-condition grows, exacerbating its sensitivity to roundoff.

Why must A become more ill-conditioned?

Matrix A approximates the unbounded differential operator $\mathrm{div}(p\cdot\mathbf{grad}(\ldots)) + q$.
Smaller singular values of A approximate those of the differential operator, but

$$\|A\| \to \infty , \text{ typically like } 1/\theta^2 .$$

How to attenuate ill effects of A's ill-condition upon the solution of "$A\cdot\boldsymbol{f} = \mathbf{b}$" :

Iterative Refinement:

Let $\mathbf{f}$:= computed value of $\boldsymbol{f}$ in "$A\cdot\boldsymbol{f} = \mathbf{b}$", and let $\mathbf{f}$'s ***Residual*** be

$$\mathbf{r} := A\cdot\mathbf{f} - \mathbf{b} .$$

Let Δ := computed value of $\Delta\boldsymbol{f}$ in "$A\cdot\Delta\boldsymbol{f} = \mathbf{r}$" as $\mathbf{f}$ was computed but faster.

Then, provided $\mathbf{r}$ was computed ***accurately enough***,

$$\mathbf{f} - \Delta\mathbf{f} \approx \boldsymbol{f} \text{ rather better than } \mathbf{f} \text{ did.}$$

• How accurate a residual $\mathbf{r}$ is "***accurately enough***" ?

"$\mathbf{r} := A\cdot\mathbf{f} - \mathbf{b}$" must be accumulated extra-precisely lest it drown in its own roundoff; and if stored arrays A and **b** were rounded off, re-compute them extra-precisely too.

Otherwise, though its residual may become smaller, $\mathbf{f} - \Delta\mathbf{f}$ can be less accurate than $\mathbf{f}$.

• What if extra-precise arithmetic is unavailable or too slow?

Then a trick must be used to compute residual $\mathbf{r}$ accurately enough.

Let's see how the trick works on a concrete example …

The regular solutions $u(\mathrm{x})$ of the singular differential equation

$$(p\cdot u')' + q\cdot u \;:=\; (\mathrm{x}\cdot u'\,)' + 4\mathrm{x}\cdot(1-\mathrm{x}^2)\cdot u = 0$$

all have $u'(0) = 0$ and so $u(-\mathrm{x}) \equiv u(\mathrm{x})$. We wish to compute a regular solution satisfying the boundary conditions $u(\pm 1) = 1$ as if we did not know that $u(\mathrm{x}) = \exp(1-\mathrm{x}^2)$. The computation will be complicated by the differential equation's singular solutions

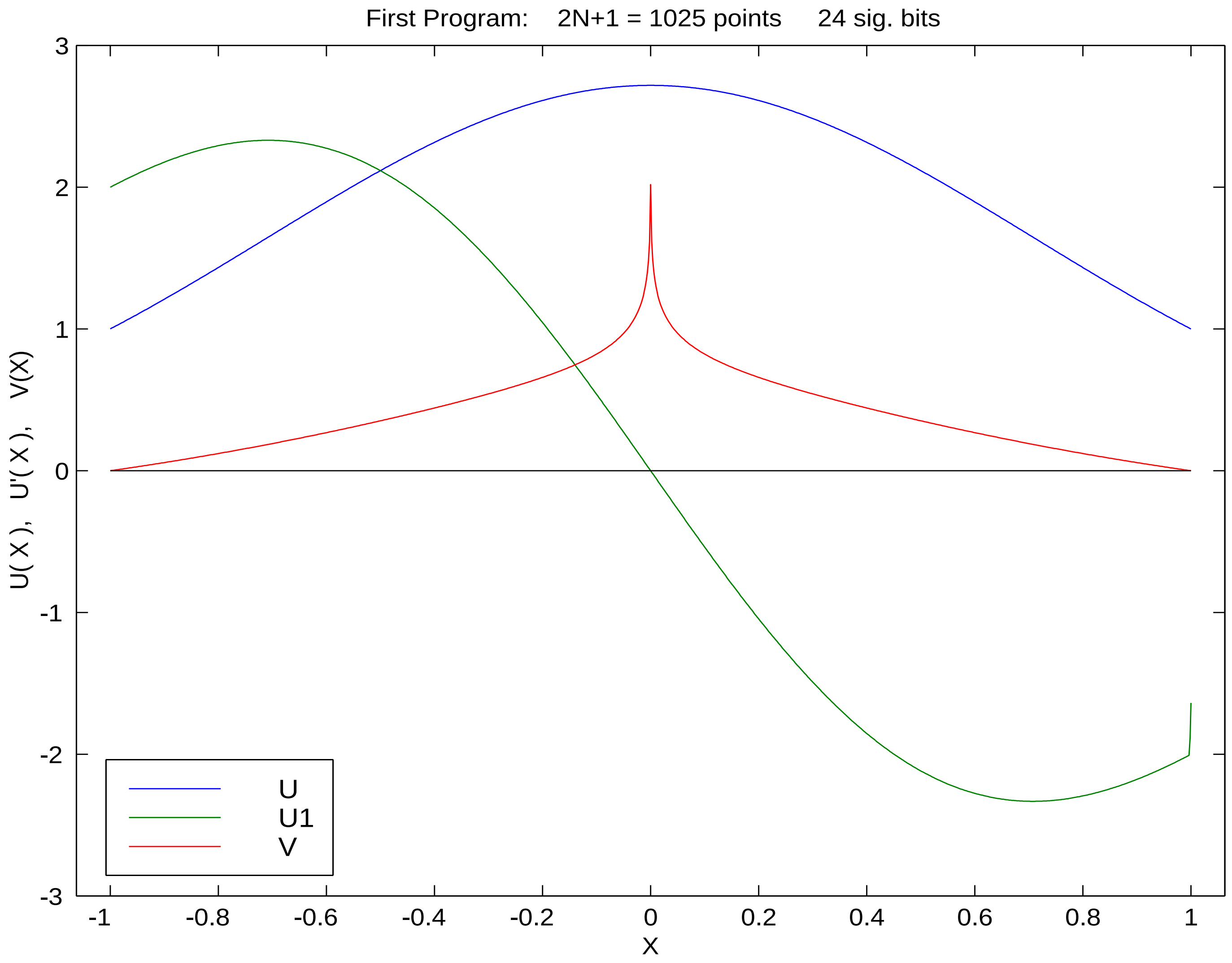
$$v(\mathrm{x}) := \mathrm{C}\cdot\exp(-\mathrm{x}^2)\cdot\int \exp(2\mathrm{x}^2)\cdot d\mathrm{x}/\mathrm{x} \;=\; \mathrm{C}\cdot\exp(-\mathrm{x}^2)\cdot\Big(\ln(|\mathrm{x}|) - \int_{|\mathrm{x}|}^{1} (\exp(2\xi^2) - 1)\cdot d\xi/\xi\,\Big)\ .$$

Their constants C can be different for $\mathrm{x} > 0$ than for $\mathrm{x} < 0$. All have a logarithmic pole at $\mathrm{x} = 0$. The pole can amplify tiny perturbations of the differential equation into a narrow spike at $\mathrm{x} = 0$. Worse, this singular solution v satisfies $v(-\mathrm{x}) \equiv v(\mathrm{x})$ and $v(\pm 1) = 0$, and the differential equation except at $\mathrm{x} = 0$, so a discretized analog of $v(\mathrm{x})$ can contaminate a numerical approximation of a regular solution $u(\mathrm{x})$ unless filtered out.

Also estimated will be $u'(\mathrm{x}) \approx \mathrm{u}^{\ddagger}(\mathrm{x}) := (\mathrm{u}(\mathrm{x}+\theta) - \mathrm{u}(\mathrm{x}-\theta))/(2\theta) = \mathrm{u}'(\mathrm{x}) + O(\theta^2)$.

Graphs of $u(\mathrm{x})$, $u'(\mathrm{x})$ and $v(\mathrm{x})$ as computed by a first crude numerical program are plotted on the next page. What caused the spike at the end of the graph of $\mathbf{u}^{\ddagger} \approx u'$?

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



The spike in $\mathbf{u}^{\ddagger}$ was caused by roundoff; see p. 13 of `…/Math128/FloTrik.pdf` .

If a numerical solution exhibits a spike, is it due to roundoff? … to a singularity? See `…/Math128/FloTrik.pdf` for both kinds, and how they were removed Here we infer $\lim_{x\to 0} u'(x)/x = u''(0) = -2u(0)$ from the differential equation to impose an internal boundary condition that filters $v(x)$ out. Then $u(x) = u(-x)$ need be computed only for $-1 \le x \le 0$.

Choose integer $N >> 2$, and set grid-gap $\theta := 1/N$ and grid-points $x_j := j\cdot\theta - 1$ for $j = 0, 1, 2, \dots, N-1, N$. Now $u(x_j)$ will be approximated by element $\mathbf{u}_j$ of a column $\mathbf{u}$ satisfying a linear system $(T + \mathrm{Diag}(\mathbf{q}))\cdot\mathbf{u} = \mathbf{r}$ with discretization error $O(\theta^2)$. The elements of N-by-N symmetric tridiagonal T, $\mathbf{q}$ and $\mathbf{r}$ are provided on p. 7 and p. 11 of `…/FloTrik.pdf` . (Here p.7's $A \Leftrightarrow T + \mathrm{Diag}(\mathbf{q})$, $\boldsymbol{f} \Leftrightarrow \mathbf{u}$, $\mathbf{b} \Leftrightarrow \mathbf{r}$.)

Computed too is column $\mathbf{u}^\ddagger$ whose elements $\mathbf{u}^\ddagger_j$ approximate the gradient $u'(x_j) \approx \mathbf{u}^\ddagger_j := (\mathbf{u}_{j+1} - \mathbf{u}_{j-1})/(2\theta)$ with error $\boldsymbol{O}(\theta^2)$; *cf*. pp. 11 & 25 of `…/FloTrik.pdf` .

A program that used Gaussian Elimination (triangular factorization into bidiagonal factors) in `float` arithmetic (24 sig.bits) gave results tabulated on the next page …

Results from a program carrying 24 sig. bits ($\varepsilon \approx 6/10^8$)

N	err(**u**)	err(**u**)·N²	err(**u**‡)	err(**u**‡)·N²
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

$N = \#\text{gaps} = 1/\theta$, $\mathrm{err}(\mathbf{u}) := \max_j |\mathbf{u}_j - u(x_j)|$, $\mathrm{err}(\mathbf{u}^\ddagger) := \max_j |\mathbf{u}^\ddagger{}_j - u'(x_j)|$.

When $N = 1/\theta$ gets too big, error worsens. The accuracy of **u** never gets much better than half the digits carried; and the accuracy of $\mathbf{u}^\ddagger$ fluctuates in a way that undermines confidence only because we know what the correct values should be.

Iterative Refinement requires the computation of residuals $\mathbf{s}_j$ from formulas like …

“ $\mathbf{s}_j := \mathbf{r}_j - a_{j-1}\cdot\mathbf{u}_{j-1} - g_j\cdot\mathbf{u}_j - a_j\cdot\mathbf{u}_{j+1}$ ” (Crude residual)

in which a_{j-1} , a_j and $g_j := \mathbf{q}_j - a_{j-1} - a_j$ are coefficients in a row of $T + \mathrm{Diag}(q)$. But g_j is ***rounded*** so, even if $\mathbf{s}_j$ is accumulated extra-precisely, iterative refinement never improves the accuracies of $\mathbf{u}$ and $\mathbf{u}^{‡}$ much.

The **trick** evaluates a more accurate residual this way instead: (no g_j)

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

HONOR PARENTHESES !

This expression takes advantage of exact cancellation among differences between nearby floating-point values of slowly varying functions. Cost: five extra subtractions.

Results from M passes of iterative refinement using the tricky formula are exhibited on the next page.

•••••••µ••••••••••••••••••••••••••

(Yes, a simpler way to solve this example’s boundary-value problem is a *Shooting Method* which converts the problem into an initial-value problem. That is the first example on p. 6 ; its $v(\tau)/v(1) = u(-\tau)$ here. But no shooting method will work on the example after this one.)

Results from a program carrying 24 sig. bits ($\varepsilon \approx 6/10^8$)

N	M	err(**u**)	err(**u**)·N^2	err(**u**‡)	err(**u**‡)·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

N = #gaps = $1/\theta$, M = #refinements, err(**u**) := $\max_j |\mathbf{u}_j - u(x_j)|$, err(**u**‡) := $\max_j |\mathbf{u}^‡_j - u'(x_j)|$.

Iterative refinement with an accurate residual added one to three sig.dec. to $\mathbf{u}$ and $\mathbf{u}^{\ddagger}$.

Why do we care about the accuracy of the gradient approximated by $\mathbf{u}^{\ddagger}$?

“No man is an *Island*, entire of itself.
… … … … And therefore
never send to know for whom the *bell* tolls;
It tolls for *thee*.”

Meditations XVII
John Donne (~1571 - 1631)

Donne’s adage for computing:

“No computation is an *Island*, entire of itself.”
It is always a means to some other end,
often via more computation.

The solutions of boundary-value problems are followed by estimates of gradients that represent strain caused by loads, electric field intensity, velocity of fluid flow, … .

Two-Dimensional Boundary-Value Problem: Laplace's Equation

$\Phi(x, y)$ is given on the boundary $\partial\Omega$ of a unit square $[0, 0] \le [x, y] \le [1, 1]$.

$\nabla^2\Phi := \partial^2\Phi(x, y)/\partial^2x + \partial^2\Phi(x, y)/\partial^2y = 0$ inside the square Ω .

For our numerical example we choose $-4.16 < \Phi(x, y) := \log((x+1/8)^2 + y^2) < 0.82$.

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

Discretization:

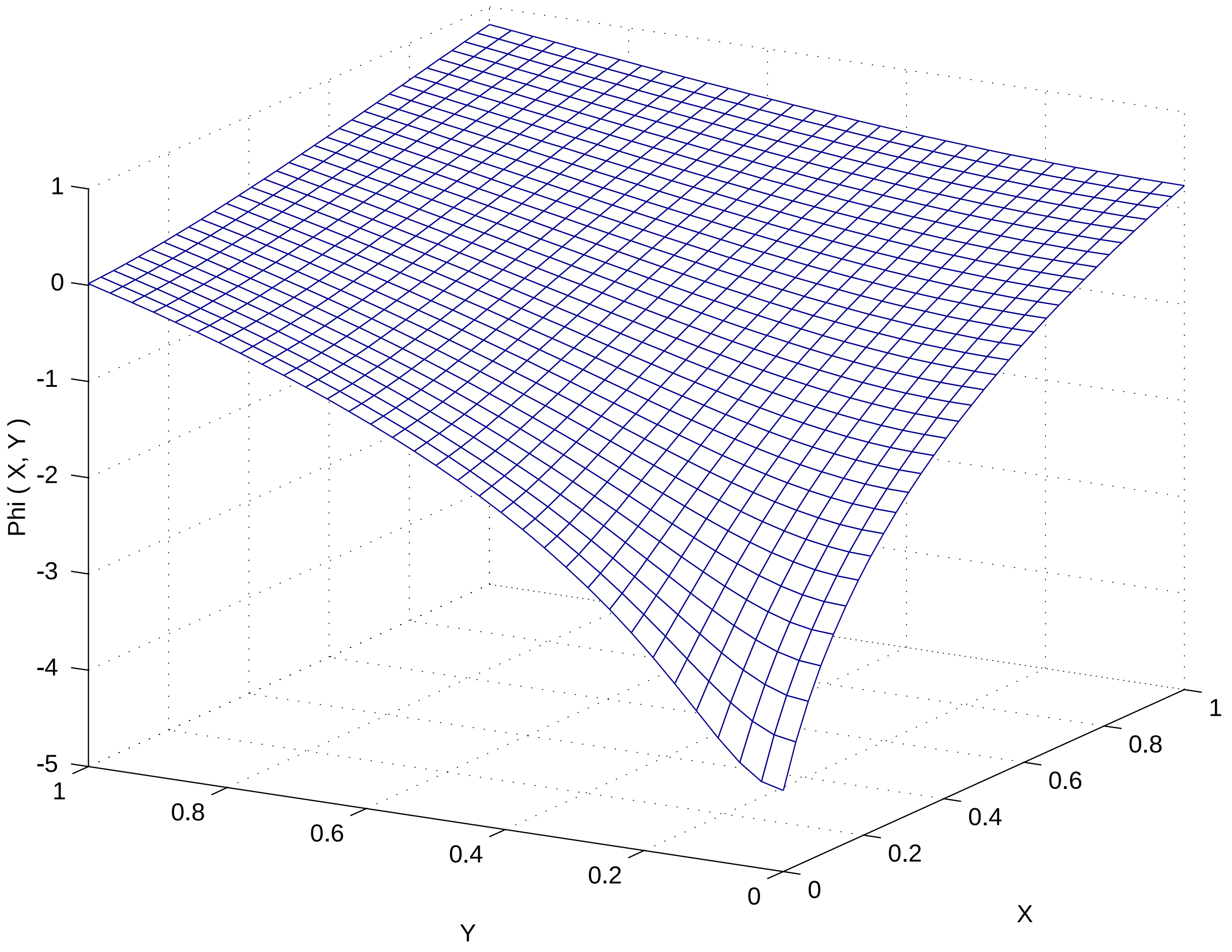
Approximate $\Phi(x, y)$ by $F(x, y)$ taking values on the intersections of a Mesh;

Mesh breaks unit square into small squares each $\theta := 1/N$ on a side.

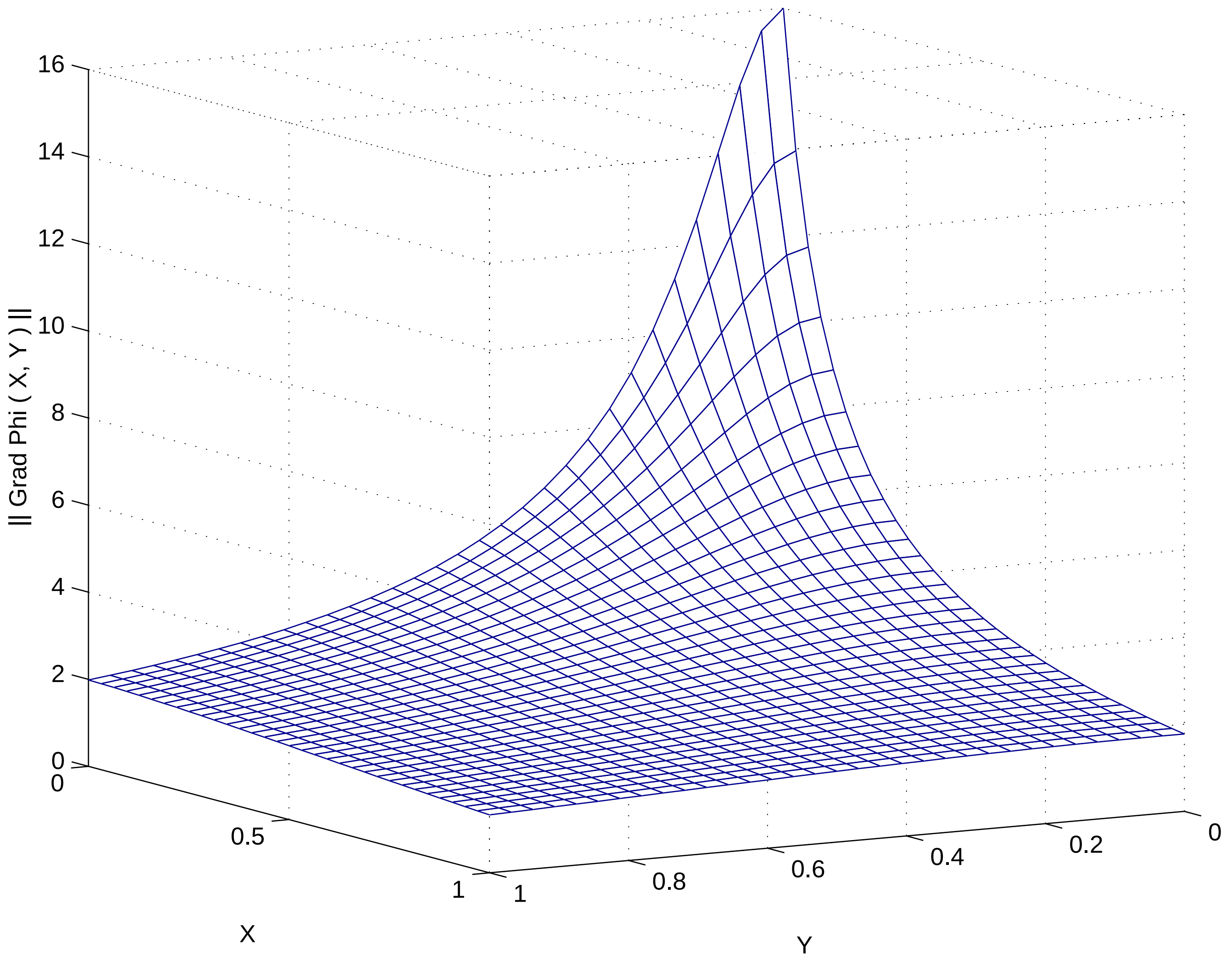
Approximate *Differential* operator $\nabla^2\Phi$ by a *Difference* operator

$$♠\Phi := \Big(\Phi(x-\theta, y) + \Phi(x+\theta, y) - 4\cdot\Phi(x, y) + \Phi(x, y-\theta) + \Phi(x, y+\theta) \Big)/\theta^2$$
$$= \nabla^2\Phi + \boldsymbol{O}(\theta^2) \text{ , so } F \text{ will satisfy } ♠F = 0 \text{ inside } \Omega \ldots \Leftrightarrow A\cdot\boldsymbol{f} = \mathbf{b} \text{ .}$$

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



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



The coordinates' origin lies behind and under the surface.

Let’s solve $A\cdot\boldsymbol{f} = \mathbf{b}$ for column $\boldsymbol{f}$ = [Values of F(x, y)] given $\mathbf{b}$ = [Boundary Values]
using `float` arithmetic (24 sig.bits) ,
and mesh-gap $\theta = 1/N$ on each of the square’s sides.

Tabulated below are the worst error difference $E := |F(x, y) - \Phi(x, y)|$, and $E\cdot N^2$,
True E from true $\boldsymbol{f}$, or *E* from *Computed* **f** ,
and under it *E* after one iterative refinement,
which used a *tricky* residual *R* , or a *crude* residual R .

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

Worse!

What is the trick ?

The crude residual was computed from the difference operator literally thus:

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

The trick computed that residual entirely in `double` arithmetic upon `float` operands, or else in `float` arithmetic thus: (Honor parentheses**!**)

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

Can you see why this trick works? Cost: three extra $\pm$, less one $\times$

It also improves the accuracy of the gradient computed from differences of F . And when " $A\cdot \boldsymbol{f} = \mathbf{b}$ " is solved by iteration the trick reduces the amplitude of *Dithering*; see `www.eecs.berkeley.edu/~wkahan/Math128/SlowIter.pdf` .

And a similar trick works for many finite-elements' difference operators.

What about the Gradient?

The simplest approximation to the derivative $\Phi'(x, y)$ is the *Central Divided Difference*

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

This will be approximated by $F^{\ddagger}(x,y)$, thus incurring error from three sources:

- $O(\theta^2)$ 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 importance of these error-sources is hardly ever knowable in advance.

The last source, roundoff, depends upon algorithmic details and tricks.

The second-last, $(F - \Phi)^{\ddagger}$, is usually much smaller than $(F - \Phi)/\theta$ because the discretization error $(F - \Phi)$ is usually smoothly *Pillow-Shaped*.

The first source, $O(\theta^2)$, overwhelmingly dominates in this example.

How can this dominance be revealed?

Compute F more accurately at larger mesh-gaps θ . …

A 6th-order (9-point) discretization of the Laplacian:

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

F was recomputed to satisfy “ $\clubsuit F = 0$ ” instead of “ $\spadesuit F = 0$ ” .

The trick to attenuate roundoff replaced the crude formula for $\clubsuit$ above by the tricky …

$$\begin{aligned}\clubsuit F(x,y) := \big(\ 4\cdot(\theta^2\cdot\spadesuit F(x,y)) + \big(((F(x-\theta,y+\theta) - F(x,y)) + (F(x+\theta,y-\theta) - F(x,y)))\ + \\ + ((F(x-\theta,y-\theta) - F(x,y)) + (F(x+\theta,y+\theta) - F(x,y))))\ \big)/(6\cdot\theta^2)\ .\end{aligned}$$

Computed Errors $E := \max_{x,y}|\,F(x,y) - \mu(x,y)\,|$ are tabulated on the next page:

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.
E_{true2}	E computed from the tricky formula for ♠F carrying 53 sig.bits.

This last E_{true2} imakes it easier to compare convergence rates of ♠ and ♣ .

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			9.681e-5	9.705e-5	
	9.677e-5	1.6e3	9.681e-5	9.658e-5	4.387e-3
32			2.039e-6	2.039e-6	
	2.084e-6	2.2e3	2.039e-6	2.039e-6	1.179e-3
64			6.928e-7	9.254e-6	
	3.225e-8	2.2e3	1.708e-7	1.635e-6	2.979e-4
128			1.859e-6	2.760e-6	
	5.126e-10	2.3e3	2.845e-7	2.290e-6	7.481e-5
256	8.103e-12	2.3e3	5.362e-7	2.366e-6	
	8.104e-12	2.3e3	2.283e-7	2.720e-6	1.872e-5
512	1.121e-13	2.0e3	3.665e-7	2.685e-6	
	1.266e-13	2.3e3	2.900e-7	8.151e-6	4.681e-6

E_{trick6} is already about as accurate as possible in 24 sig.bits when $N \ge 64$.

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

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.

N	D_{true6}	$\ldots\cdot N^2$	D_{trick6}	D_{crude6}	D_{trick2}
16			0.3207	0.3207	0.2948
	0.3207	82.09	0.3207	0.3207	0.2948
32			0.161	0.161	0.1537
	0.161	164.8	0.161	0.161	0.1537
64			0.05772	0.05767	0.05467
	0.05773	236.5	0.05773	0.05772	0.05458
128			0.01732	0.0174	0.01593
	0.01731	283.6	0.01731	0.01732	0.0161
256			0.004778	0.004847	0.004456
	0.004745	311	0.004727	0.004847	0.004415
512			0.001297	0.001398	0.001328
	0.001243	325.8	0.001297	0.001398	0.001231

Tabulated errors $F^{\ddagger} - \Phi'$ reflect a contribution roughly $333\cdot\theta^2$ due mostly to either a mesh-gap θ too big, or a 2nd-order formula $F^{\ddagger}$ too crude, rather than big $F - \Phi$.

The accuracy of this example's computed F is overkill for the gradient unless that accuracy is exploited by higher-order divided difference approximations to derivatives.

Higher-order formulas for discretized first derivatives:

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

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

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

Complication: The analogous formulas to estimate $\Phi'(x, y)$ from F differ according to how close (x, y) is to the boundary $\partial\Omega$ of the square.

This complication does not alter our tricks.

Conclusions:

- Aided by tricks, 4-byte `floats` are accurate enough for many differential equations.
- Otherwise, `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.
- Only if residuals are computed accurately enough must iterative refinement enhance accuracy after discretization is refined by an increase in the density of mesh-points.
  But the *Law of Diminishing Returns* cannot be postponed forever.
- Tricks are palliatives, not cures for ailments that afflict scientific and engineering computations now that floating-point is optimized for entertainment.

Ailments?

- Inadequate tools to help diagnose bugs peculiar to floating-point's roundoff *etc*.
- Widespread misunderstandings of roundoff among most scientists and engineers,
  especially among Computer Scientists.

  Consequently programming languages become ever less hospitable to numerically naive but otherwise clever programmers who use floating-point only occasionally.