15. Stability analysis and computer experiment:

Test the stability of the scheme from problem 13 for $d = 0$:

$$\frac{0.5T_j^{n-1} - 2T_j^n + 1.5T_j^{n+1}}{\Delta t} - \frac{\alpha(T_{j-1} - 2T_j + T_{j+1})^n}{\Delta x^2} = 0 \tag{1}$$

with the von Neumann method. (Hint: This requires the solution of a quadratic equation in the amplification factor g. You can abbreviate $s\sin^2(k\Delta x/2)$ with h to simplify the expression. Discuss separately the cases for which g has real and complex solutions).

Solution:

Von Neumann method:

$$T_j^n = \widetilde{T}\exp\left[i(\omega t_n + kx_j)\right] \tag{2}$$

such that $T_j^{n+1} = T_j^n \exp[i\omega\Delta t]$ and $T_{j+1}^n = T_j^n \exp[ik\Delta x]$. Here it is assumed that the equation is linearized (this yields the same form as the heat conduction equation because it is already a linear equation for $\alpha = const$). The amplification factor is defined as $g = T_j^{n+1}/T_j^n = \exp[i\omega\Delta t]$ and stability requires $|g| \leq 1$. Note that g is in general in the complex plane! Substituting (2) into (1) yields:

$$\left(\frac{1}{2\Delta t}\exp[-i\omega\Delta t] - \frac{2}{\Delta t} + \frac{3}{2\Delta t}\exp[i\omega\Delta t]\right)T_j^n - \frac{\alpha}{\Delta x^2}\left(\exp[-ik\Delta x] - 2 + \exp[ik\Delta x]\right)T_j^n = 0$$

Multiplying with Δt, using the definition for g, and noting that
$\cos(k\Delta x) = 0.5\left(\exp[-ik\Delta x] + \exp[ik\Delta x]\right)$ we obtain

$$\frac{1}{2g} - 2 + \frac{3}{2}g - 2s(\cos(k\Delta x) - 1) = 0$$

With the relation $1 - \cos(k\Delta x) = 2\sin^2(k\Delta x/2)$ and multiplying with $2g/3$ we obtain

$$\begin{aligned} g^2 - \frac{4}{3}(1-2h)g + \frac{1}{3} &= 0 \\ with\ h &= s\sin^2(k\Delta x/2) \end{aligned}$$

with the roots:

$$g = \frac{2}{3}(1-2h) \pm \frac{1}{3}\sqrt{4(1-2h)^2 - 3}$$

To determine the amplitude of g we need to know the range of h for which g is real or in the complex plane.

i) g is complex if $4(1-2h)^2 - 3 < 0$ or

$$(1-2h)^2 < \tfrac{3}{4} \quad or \quad \begin{array}{lll} 1-2h < \frac{1}{2}\sqrt{3} & => & h > \frac{1}{2} - \frac{1}{4}\sqrt{3} \\ 2h-1 < \frac{1}{2}\sqrt{3} & => & h < \frac{1}{2} + \frac{1}{4}\sqrt{3} \end{array}$$

In this case we obtain $|g|^2 = gg^* = \frac{4}{9}(1-2h)^2 + \frac{1}{9}(3 - 4(1-2h)^2) = \frac{1}{3}$ such that the code is stable in the range:

$$\tfrac{1}{2} - \tfrac{1}{4}\sqrt{3} < s\sin^2(k\Delta x/2) < \tfrac{1}{2} + \tfrac{1}{4}\sqrt{3}.$$

ii) For stability, the maximum of g must not exceed 1:

$$\tfrac{2}{3}(1-2h)+\tfrac{1}{3}\sqrt{4(1-2h)^2-3}\leq 1 \text{ or } \tfrac{1}{9}(4(1-2h)^2-3)\leq\left[1-\tfrac{2}{3}(1-2h)\right]^2$$

which yields $h\geq 0$ or $s\sin^2(k\Delta x/2)\geq 0$ which is always satisfied.
iii) Similarly the minimum of g must not be smaller than -1 for stability. Thus

$$\tfrac{2}{3}(1-2h)-\tfrac{1}{3}\sqrt{4(1-2h)^2-3}\geq -1 \text{ or } \left[1+\tfrac{2}{3}(1-2h)\right]^2\geq\tfrac{1}{9}(4(1-2h)^2-3)$$

which yields $h\leq 1$ or $s\sin^2(k\Delta x/2)<1$ Since the argument of the sin depends on the considered wavelength, the stability limit is: $s<1$
In conclusion the modified scheme using (1) is stable in the range of $s\in[0,1]$.

16. Implementation of a different integration scheme:

Modify the program sim1 to introduce the scheme from problem 13 in place of the FTCS (forward time, centered space) scheme:

For the very first time step use the formula $\partial T/\partial t \approx (T^{n+1} - T^n)/\Delta t$ (because $T^{n-1}$is not available at $t = 0$.)

Solution:

$$\frac{0.5T_j^{n-1} - 2T_j^n + 1.5T_j^{n+1}}{\Delta t} - \alpha \left[\frac{(1+d)(T_{j-1} - 2T_j + T_{j+1})^n}{\Delta x^2} - \frac{d(T_{j-1} - 2T_j + T_{j+1})^{n-1}}{\Delta x^2} \right] = 0$$

$$\frac{3}{2}T_j^{n+1} = 2T_j^n - \frac{1}{2}T_j^{n-1} + s(1+d)\left(T_{j-1}^n - 2T_j^n + T_{j+1}^n\right) - sd\left(T_{j-1}^{n-1} - 2T_j^{n-1} + T_{j+1}^{n-1}\right)$$

or

$$T_j^{n+1} = \left[\frac{4}{3} - \frac{4}{3}s(1+d)\right]T_j^n + \frac{2}{3}s(1+d)\left(T_{j-1}^n + T_{j+1}^n\right) + \left[-\frac{1}{3} + \frac{4}{3}sd\right]T_j^{n-1} - \frac{2}{3}sd\left(T_{j-1}^{n-1} + T_{j+1}^{n-1}\right)$$

defining the coefficients as

$$s_{10} = \frac{2}{3}s(1+d) \qquad s_{00} = \frac{4}{3} - 2s_{10}$$
$$s_{11} = -\frac{2}{3}sd \qquad s_{01} = -\frac{1}{3} - 2s_{11}$$

(note the definitions of $s_{00}$and s_{01} are written to reduce the number of operations for the FORTRAN code) the scheme to update T should read

$$T_j^{n+1} = s_{00}T_j^n + s_{10}\left(T_{j-1}^n + T_{j+1}^n\right) + s_{01}T_j^{n-1} + s_{11}\left(T_{j-1}^{n-1} + T_{j+1}^{n-1}\right)$$

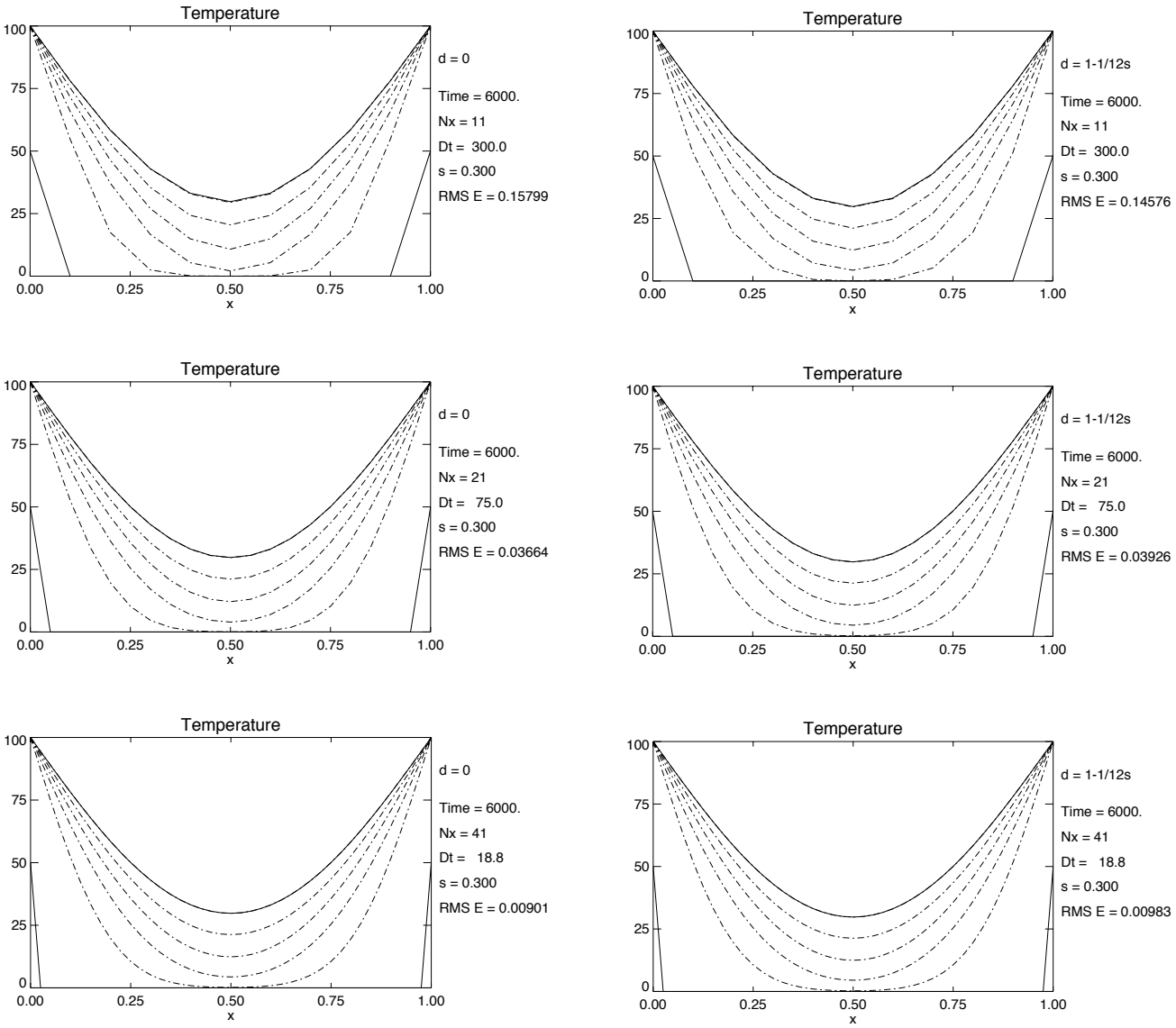
The corresponding change is found in the subroutine int2l in the new program sim12lev.f.

The tasks to modify the code should include:

- introduce the new parameter d in the sim1.dat
- declare this parameter (also in a common block) in the include file
- read this parameter in pread
- write parameter in pwrite and change write format
- introduce new integration subroutine (here called *int2l*) for the above equation (requires a new array, e.g. foold, to store data from T^{n-1})
- modify the main program such that for the first integration step the *intfts* subroutine is called and for all subsequent steps the new integration routine is called.

(a) Obtain solutions with $\Delta x = 0.1,\ 0.05,\ 0.025$ for $s = 0.3$ using $d = 0$ and $d = 1 - \frac{1}{12s}$, determine the error, and compare with the solution produced by the FTCS scheme.

$d = 0$ $\qquad\qquad$ $d = 1 - 1/12s$



(b) Is the accuracy for decreasing Δx consistent with your truncation error for problem 13? If not, why not?

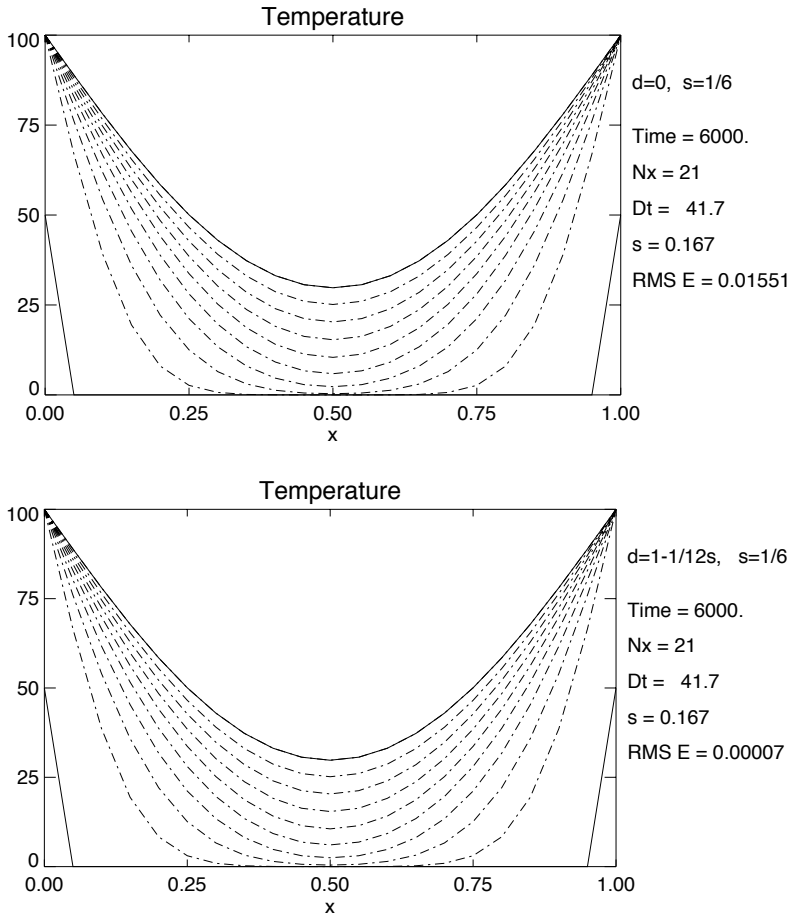
The rms error for these cases is listed in the following table:

Case	$d = 0$	$d = 1 - \frac{1}{12s}$
$\Delta x = 0.1$	0.1580	0.1458
$\Delta x = 0.05$	0.0366	0.0393
$\Delta x = 0.025$	0.0090	0.0098

As can be seen the error for $d = 0$ decreases with a factor of $1/4$ consistent with a scaling of Δx^2. However, the error for the cases with $d = 1 - \frac{1}{12s}$ also scales approximately with $1/4$ corresponding to a

scaling of Δx^2 although the error analysis from problem 13 implies that the error should scale with Δx^4. There appears to be some inconsistency or possibly even an error in the program?

The resolution of this inconsistency is indicated in the following results for $\Delta x = 0.05$, $s = 1/6$ and



With $d = 0$ for the top plot and $d = 1 - \frac{1}{12s}$ in the bottom plot. For the top case the error is 0.0155 which is fairly consistent with the values in the table for $\Delta x = 0.05$. However, for the second plot with $d = 1 - \frac{1}{12s}$ the error is 0.00007 which is much lower than the the case with the same $s = 1/6$ but $d = 0$. This indicates that this is indeed showing an error of order Δx^4. So why does this scaling show up here but not for any of the cases with $s = 0.3$.

The resolution to this puzzle is the following. The modified code uses the FCTS scheme for the very first time step. With this 1st step an error is introduced which corresponds to that of the FCTS scheme, ie., $O\left(\Delta x^2\right)$ for all values of s except for $s = 1/6$ (for which the error is $O\left(\Delta x^4\right)$). In the subsequent evolution this error is carried over because the error evolves according to the same diffusion equation as the main solution. Thus if the initial error is only 2nd order then the subsequent error is also 2nd order (until it is diminished sufficiently due to diffusion). Thus for the limited number of time steps the error remains second order even though all integration steps except for the first are 4th oder accurate. However, if the first step is carried out with 4th order accuracy ($s = 1/6$ for the ftcs) and the subsequent steps are 4th order ($d = 1 - \frac{1}{12s}$) then the error at any point in the simulation is 4th order. This is demonstrated by the last two plots. The combination $d = 1 - \frac{1}{12s}$ and $s = 1/6$ makes both integrations 4th order and the error is consistently small. The combination $s = 1/6$ and $d = 0$ makes only the first integration step 4th order. Similarly the cases with $s = 0.3$ and $d = 1 - \frac{1}{12s}$ make only the subsequent steps 4th order and the error is 2nd order.

17. How fast is your computer?

Solution:

Evaluation of computer speed in terms of floating point operations.

	Operation	No Opt	-O2	-O3
1	do	2.83	0.001	0.001
2	$a = a + b$	8.53	8.51	8.55
3	$a = a - b$	8.51	8.52	8.54
4	$a = a \cdot b$	14.20	14.20	14.23
5	$a = a/b$	36.94	36.98	37.20
6	$a = (a+b) \cdot c$	22.70	22.70	22.70
7	$a = (a \cdot b + d) \cdot c$	36.79	36.87	36.90
8	$a = (a \cdot b + d)/c$	59.54	59.57	59.80
9	$a = i + j$	8.57	8.50	8.50
10	$a = b \cdot i + j$	17.00	17.00	17.00
11	$a = i + b \cdot j$	31.40	31.27	31.26
12	$a = i + j/a$	48.40	54.10	54.20
13	$a = (i + j/a)^b$	354.00	362.50	361.70
14	$a = \sqrt{i + j/a}$	95.00	84.60	84.50
15	$a = \sin(i + j/a)$	219.00	227.40	227.20
16	$a = \exp(b(i + j/a))$	212.70	221.30	221.00
17	$e = f(j) + g(j)$	11.53	5.74	3.40
18	$e = f(j) + g(j) + h(j)$	17.23	8.58	4.28
19	$e = f(j) \cdot g(j)$	8.65	5.80	3.36
20	$e = f(j) + b \cdot g(j)$	11.41	5.93	3.42
21	$e = f(j) + (i + c \cdot j) \cdot g(j)$	54.17	54.10	3.70
22	$e(j) = f(j)/e(j) + (i + b \cdot j) \cdot g(j)$	54.18	48.40	5.02
23	$e(j) = \sqrt{(f(j))} + (i + b \cdot j) \cdot g(j)$	48.70	54.30	5.01
24	$e(j) = \sin(f(j)) + (i + b \cdot j) \cdot g(j)$	159.30	128.00	128.00
25	$e(j) = (\exp f(j)) + (i + b \cdot j) \cdot g(j)$	125.80	111.10	112.50

The table above shows timing results in seconds for 10^{10} executions of the corresponding operation on a Mac Book Pro with a 2.6 GHz (3720QM) quad-core Intel Core. Results are shown for the gfortran Fortran compiler in a non-optimized mode (default for -O1) and for the best optimization levels -O2 and -O3 (for which automatic vectorization is switched on). The higher optimizations switch on vectorization basically allowing for several operations in a single processor cycle and make use of other processor specific optimizations.Note that Intel Fortran compiler has various other optimization options that deal with the evaluation of floating point operations and parallelization. The gfortran compiler has an option -Ofast which has results similar to -O3 except that even recursive addition and subtraction execute with 2.15 seconds and basic multiplication and division are at 3.6 seconds. However, most combinations of the basic operations execute with the same times as (8, 10-17) have the same results as -O3 or -O2. All of these operations execute on a single logical core (the cpu has 8 logical cores).

A value of 1 second corresponds to 10^{10} operations per second or 10 GFlops. Processor speed would imply a basic speed of $2.6 \cdot 10^9$ operations/second or approximately 4 seconds for a loop with a single floating point operations. Operations are carried out in a do loop and it is verified that the execution time

is indeed proportional to the number of executions for each operation. The initial value of the variables varies. To insure proper execution most operations are recursive (to insure that the compiled program indeed executes the operations). Variables are chosen such that the expressions remain finite in terms of the machine accuracy for 10^{10} operations. However, results can depend on the optimization. This is in particular the case when a result depends on the machine accuracy. In this case also the execution time can be influenced by issues of the internal representation of real numbers. For example the 2nd operation (addition) uses initial values of a=1.1 and b=0.0001. To avoid overflow the value of a is divided by 99.9 (which is used only once in every 100000 operations).

Relative timing results, i.e., results are normalized to the basic '$+$' and '$-$' operations are in the table at the end.

A few things to note are

- Most operations in the first group are recursive to insure execution; However, short recursive operations also prevent vectorization or parallelization of the loop.
- Basic execution speed is about 1.2 GFlops for for addition and subtraction; Multiplication takes almost twice as long and division about 4 times as long.
- Combinations of basic recursive operations (6 to 8) are consistent with basic operations
- Basic integer operations and combinations lead to the following conclusions: The addition of 2 integers (9) takes very little time but the conversion to a real takes about 8.4 seconds. (10) requires 1 conversion and one addition of 2 reals ($b \cdot i$ is computed outside of the j loop such that it does almost not contribute to execution time). (11) requires a real conversion, a real multiplication and a real addition consistent with the execution time. (12) is consistent with (11) in replacing the multiplication with a division.
- For each of the special function calls (13-16) one has to subtract about 54 seconds (62 seconds for 16) to obtain the basic execution time for the respective function. Particularly take the power of any number is about 36 times slower than basic operations. The sqrt is about 6 times slower.
- All array operations (17-25) are much faster than expected from the basic exection times. Specifically (17) is about 2.5 times and (21) is about 15 times faster and typical speed-up of 10 are frequent for array operations.
- The speedup for special functions is only moderate except for the sqrt (23) which is much faster than in the recursive exection.
- In brief :
  - multiplications are moderately slower (20 - 30%) than the basic $+$and $-$operations;
  - divisions are by a factor of 4 slower than basic the operations (exception single division (5)).
  - intrinsic function calls are very slow.
  - array operations are much faster than recursive single number operations.
  - typical floating point operations are about 1 GFLOPS, however they can speed up to close to 10 GFLOPS implying a full speed of the processor of between 8 and 80 GLOPS

The following table show relative timing results, i.e., results are normalized to the basic '+' and '−' operations.

	Operation	No Opt	-O2	-O3
1	do	0.33	0.00	0.00
2	$a = a + b$	1.00	1.00	1.00
3	$a = a - b$	1.00	1.00	1.00
4	$a = a \cdot b$	1.67	1.67	1.67
5	$a = a / b$	4.33	4.34	4.36
6	$a = (a + b) \cdot c$	2.66	2.66	2.66
7	$a = (a \cdot b + d) \cdot c$	4.32	4.32	4.33
8	$a = (a \cdot b + d) / c$	6.98	6.99	7.01
9	$a = i + j$	1.01	1.00	1.00
10	$a = b \cdot i + j$	1.99	1.99	1.99
11	$a = i + b \cdot j$	3.68	3.67	3.67
12	$a = i + j / a$	5.68	6.35	6.36
13	$a = (i + j / a)^b$	41.52	42.52	42.43
14	$a = \sqrt{i + j / a}$	11.14	9.92	9.91
15	$a = \sin(i + j / a)$	25.69	26.67	26.65
16	$a = \exp(b(i + j / a))$	24.95	25.96	25.92
17	$e = f(j) + g(j)$	1.35	0.67	0.40
18	$e = f(j) + g(j) + h(j)$	2.02	1.01	0.50
19	$e = f(j) \cdot g(j)$	1.01	0.68	0.39
20	$e = f(j) + b \cdot g(j)$	1.34	0.70	0.40
21	$e = f(j) + (i + c \cdot j) \cdot g(j)$	6.35	6.35	0.43
22	$e(j) = f(j) / e(j) + (i + b \cdot j) \cdot g(j)$	6.36	5.68	0.59
23	$e(j) = \sqrt{(f(j))} + (i + b \cdot j) \cdot g(j)$	5.71	6.37	0.59
24	$e(j) = \sin(f(j)) + (i + b \cdot j) \cdot g(j)$	18.69	15.01	15.01
25	$e(j) = (\exp f(j)) + (i + b \cdot j) \cdot g(j)$	14.76	13.03	13.20