Resources

For people not so familiar with `python3` and `numpy` there is:

(https://numpy.org/doc/stable/user/quickstart.html)

Numerical exercises

Exercise 1.1:

1.) Suppose $u \in C^\infty([a, b])$. As the hint suggests we use Taylor series about x_0 and evaluate at $x_0 + k\Delta x$:

$$u(x_0 + k\Delta x) = u(x_0) + (k\Delta x)u'(x_0) + \frac{1}{2}(k\Delta x)^2 u''(x_0) + \frac{1}{6}(k\Delta x)^3 u^{(3)}(x_0) + \frac{1}{24}(k\Delta x)^4 u^{(4)}(x_0) + \cdots.$$

We want $u''(x_0)$. The idea is thus to combine pick different k and combine terms. In particular for $k \in \{-1, 0, 1\}$:

$$\begin{aligned} u(x_0 - \Delta x) &= u(x_0) - (k\Delta x)u'(x_0) + \frac{1}{2}(k\Delta x)^2 u''(x_0) - \frac{1}{6}(k\Delta x)^3 u^{(3)}(x_0) + \cdots, \\ u(x_0) &= u(x_0), \\ u(x_0 + \Delta x) &= u(x_0) + (k\Delta x)u'(x_0) + \frac{1}{2}(k\Delta x)^2 u''(x_0) + \frac{1}{6}(k\Delta x)^3 u^{(3)}(x_0) + \cdots, \end{aligned}$$

Thus we find:

$$u(x_0 - \Delta x) - 2u(x_0) + u(x_0 + \Delta x) = \Delta x^2 u''(x_0) + \frac{1}{12}(\Delta x)^4 u^{(4)}(x_0).$$

Without loss of generality we put $x_0 = 0$, take $x_i := i\Delta x$ and use the short-hand $u_{i+k} := u((i+k)\Delta x)$. We can summarize the above as:

$$u''(x)|_{x=x_i} = \frac{1}{(\Delta x)^2}(u_{i-1} - 2u_i + u_{i+1}) + \mathcal{O}(\Delta x^2);$$

so we found a *central, second order approximation* to the second derivative.

Remark I: In a similar fashion one can construct higher-order approximants to any derivative operator we happen to be interested in. Generally this requires more points (in k) i.e., a wider stencil. We could also bias the stencil toward some direction.

Remark II: Some additional, useful stencils (verify you can derive them!):

$$\begin{aligned} u'(x)|_{x=x_i} &= \frac{1}{2\Delta x}(-u_{i-1} + u_{i+1}) + \mathcal{O}(\Delta x^2); \\ u''(x)|_{x=x_i} &= \frac{1}{(\Delta x)^2}\left(-\frac{1}{12}u_{i-2} + \frac{4}{3}u_{i-1} - \frac{5}{2}u_i + \frac{4}{3}u_{i+1} - \frac{1}{12}u_{i+2}\right) + \mathcal{O}(\Delta x^4); \end{aligned}$$

Remark III: In the case of non-uniform grids this becomes more involved.

2.) The order of convergence should match that of the underlying approximants.

Note *convergence testing*: Suppose we are solving a problem in 1^{d} over a uniform grid with spacing Δx and find a numerical solution $u_{\Delta x}(x)$. We again assume that Taylor expansion is viable and write:

$$u_{\Delta x}(x) = u(x) + \Delta x E_1 + (\Delta x)^2 E_2 + (\Delta x)^3 E_3 + \mathcal{O}((\Delta x)^4),$$

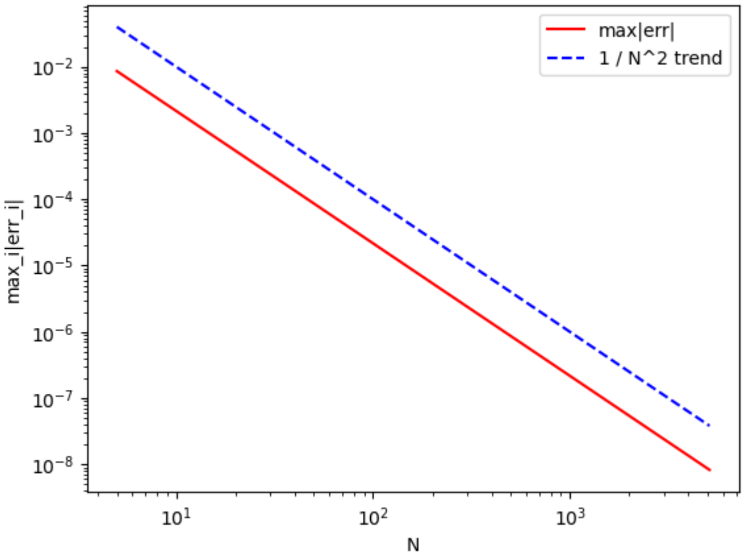
where $u(x)$ is the true solution and we have introduced error constants E_i which are assumed to be independent of the spacing Δx. Suppose we have a second order scheme and that we successively halve Δx:

$$\begin{aligned} u_{\Delta x}(x) - u(x) &= (\Delta x)^2 E_2 + (\Delta x)^3 E_3 + \mathcal{O}((\Delta x)^4), \\ 4\big(u_{\Delta x/2}(x) - u(x)\big) &= (\Delta x)^2 E_2 + \frac{(\Delta x)^3}{2} E_3 + \mathcal{O}((\Delta x)^4), \\ 16\big(u_{\Delta x/4}(x) - u(x)\big) &= (\Delta x)^2 E_2 + \frac{(\Delta x)^3}{4} E_3 + \mathcal{O}((\Delta x)^4), \\ &\vdots \\ 2^{2k}\big(u_{\Delta x/k}(x) - u(x)\big) &\to (\Delta x)^2 E_2. \end{aligned}$$

Remark: Notice that if we didn't have an analytical solution $u(x)$ we could instead consider the difference between two numerical solutions:

$$2^{2k}\left(u_{\Delta x/k}(x) - u_{\Delta x/(2k)}(x)\right) \to \frac{3}{4}(\Delta x)^2 E_2.$$

In our case we find an overall second order trend (see script):



3.) Here we want a tri-diagonal solver. There is the so-called Thomas algorithm for this, however, we can also use something provided in `scipy.linalg.solve_banded`.

Thomas algorithm:

(https://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm)

`scipy` reference document:

(https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.solve_banded.html)

Remark: The initial situation is actually even worse than the question remarks! Suppose we wanted to solve for a variety of differing inhomogeneities f, as initially implemented we would have to invert A each time.

4.) Use self-consistent convergence testing (SCCT).

Idea: Sometimes we don't have an analytical solution available and yet we still need to come up with something to test with - we can compare multiple numerical solutions against each other.

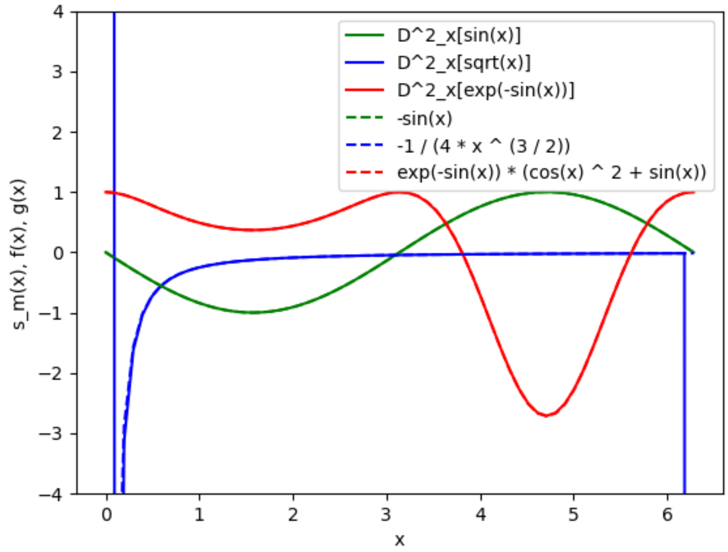
Another approach is to manufacture a solution. In the present context, the idea goes as follows:

- Insert some known u into $-u''(x) = f(x)$ thus generating $f(x)$.
- Fix this as the choice of $f(x)$ and attempt to reconstruct numerically the prescribed u; verify that anticipated convergence rates are observed.

Exercise 1.2:

1.) Notice immediately that $s_m(x)$ (for $m \in \mathbb{Z}$) and $g(x)$ satisfy the underlying periodicity of the problem whereas $f(x)$ does not.

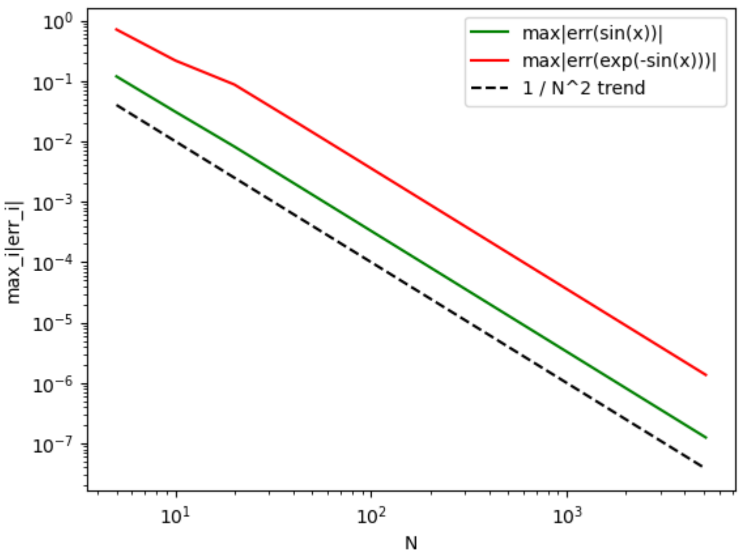
Using `waveP_1p1d.py` we can inspect the derivatives for $N = 64$, say:



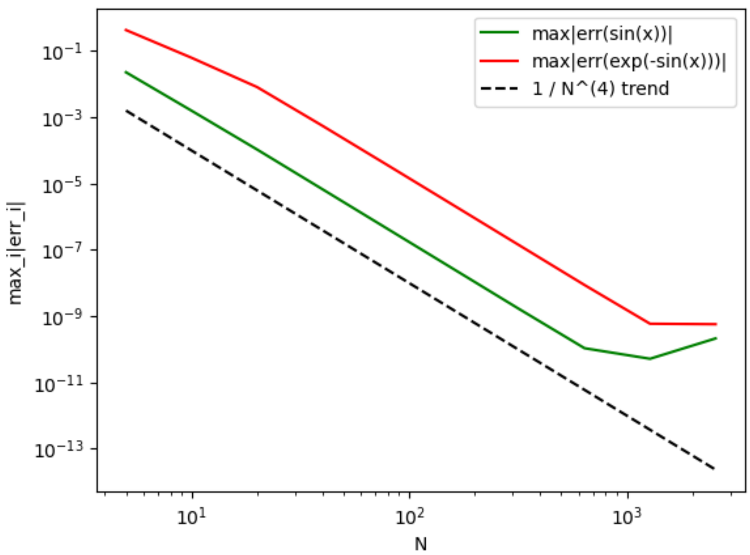
Where we make use of ghost-nodes to enforce periodicity in the sense shown in the script.

Convergence :

We can quickly inspect behaviour (using a similar approach to the Poisson problem):



If we instead use `nghost=2` together with the 4^{th} order accurate approximant from earlier then:



2.) It is a good idea to first write down the full method before trying to implement anything. The stages are given by (see Table of Eq.(6)):

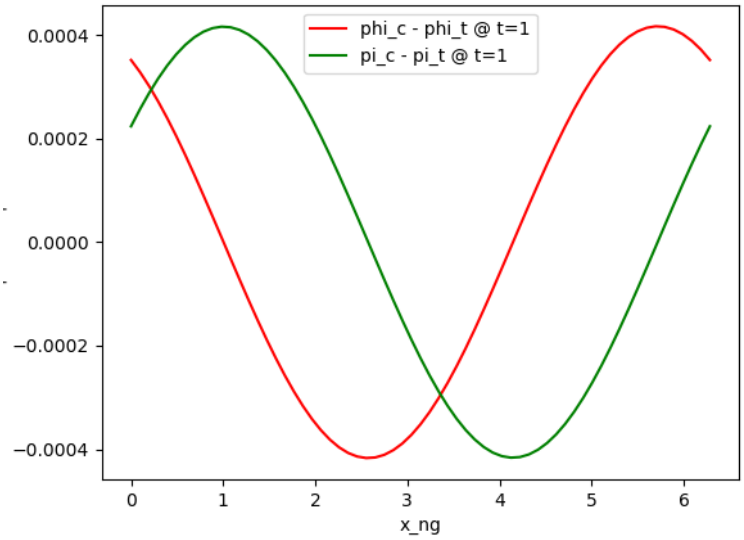
$$
\begin{aligned}
v_1 &= f(t_n,\, u_n),\\
v_2 &= f\left(t_n + \frac{1}{2}\Delta t,\, u_n + \frac{1}{2}\Delta t v_1\right),\\
v_3 &= f\left(t_n + \frac{1}{2}\Delta t,\, u_n + \frac{1}{2}\Delta t v_2\right),\\
v_4 &= f(t_n + \Delta t,\, u_n + \Delta t v_3);
\end{aligned}
$$

where the stages are assembled according to the linear combination:

$$
u_{n+1} = u_n + \Delta t\left(\frac{1}{6}v_1 + \frac{1}{3}v_2 + \frac{1}{3}v_3 + \frac{1}{6}v_4\right).
$$

Details of grid extension must also be dealt with - an example of this is in the script.

- For a concrete calculation: set $\phi_0 = -\sin(mx)$ and $\pi_0 = m\cos(mx)$ with $m = 1$. Choose `nghost=2` and $(N_t,\, N_x) = (200,\, 64)$ with $t \in [0,\, 1]$ (CFL of $\simeq 0.05$).



- Overall 4^{th} order scheme, field components on physical grid are $\mathcal{O}(1)$ we see errors $\mathcal{O}(10^{-4})$ so this suggests the method is implemented adequately.

3.) Experimenting with CFL and evolution for longer durations in time shows that unstable behaviour occurs (numerical solution spuriously explodes). In other words *if* a δt is chosen such that the numerical DOD no longer contains the true DOD of the underlying PDE we have stability problems.

4.) There are a variety of methods to construct an analytical solution:

- Separation of variables $\phi(t,\, x) := T(t)X(x)$
- Characteristics
- We could also fix a choice $\tilde{\phi}(t,\, x)$ and consider performance on the related inhomogeneous problem:

  $$
  \partial_t^2[\phi] - \partial_x^2[\phi] = f(t,\, x),
  $$

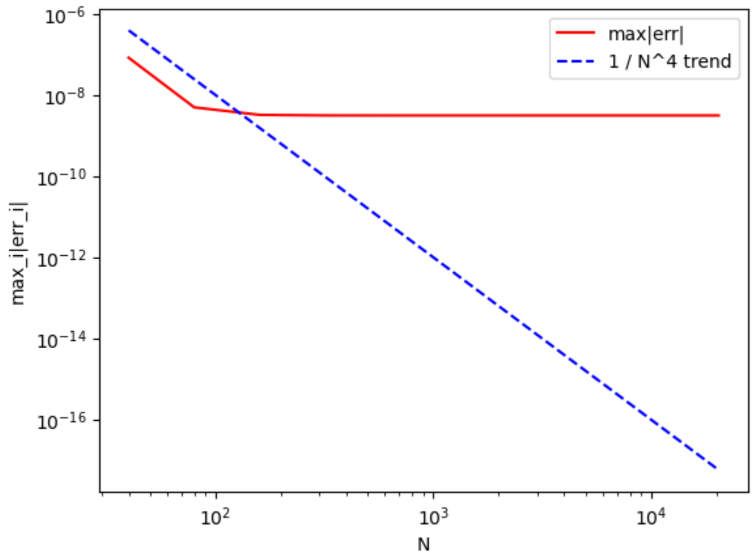
  where f can be thought of as the functional generated by $\tilde{\phi}$, i.e. $f[\tilde{\phi}] := \partial_t^2[\tilde{\phi}] - \partial_x^2[\tilde{\phi}]$.

For testing note that $\phi(t,\, x) = \sin(m(t - x))$ satisfies the original PDE with BC and is of the form $g(t - x)$. This gives a rightward propagating sinusoid.

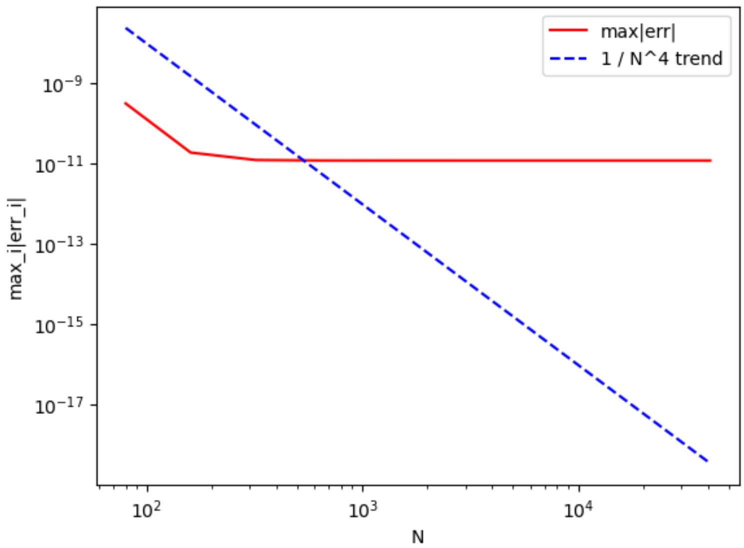
We can take again $\phi|_{t=0} = -\sin(mx)$ and $\pi|_{t=0} = m\cos(mx)$.

We compare RMS error at overall 4^{th} order:

Evolution on $t \in [0,\, 10]$ with $(N_t,\, N_x) = (40,\, 32)$:



Evolution on $t \in [0, 10]$ with $(N_t, N_x) = (80, 64)$:



Note saturation at lower overall RMS.