1 Non-linear contrubution (Bad way to do it)

Martin: We can look at k =10
Just note that, if we dont run the GEvolution from enough large scale, we lose the the long mode modulation and the pertubration on quadratic variables would change like χ .., and on the other hand also we have the large power from non linearities of scalar field Here we write the full equations and try to solve them numerically in Gevolution, and if we got something intreesting or strange we need to solve them in mathematica to see if we did not have make a mistake !

To solve in mathematica we must solve with Non -linear solve command and to exactly compare with Gevolution we can get some symmetric situations. The discussion with Julian is as following,

> I just have a problem with the last part of the note where the time solution of discrete Poisson equation is written
> It looked strange to me that for any modes the time dependence of the growing mode decreases!

No, epsilon can have either sign. It vanishes for K=Laplace which is the continuum limit. For K>Laplace epsilon is negative, while it is positive for K<Laplace

> So I solved the Poisson equation myself and took
> the real part, the solution in mathematica suggest different relation
> than is written in the note. At the end depending on the K and Delta we
> get different behaviour for growing mode. Did I make a mistake somewhere?

Please simplify your expressions and you'll see that you got exactly what I wrote.

> P_{\phi'\cH}(k) ~ P_{\phi}(k) * 6.675 * (k/k_Ny)^4
>
> where k_Ny is the Nyqvist wavenumber in your simulation
> Can you please give me more information about where this formula come from?

The first (and least trivial) step is to compute the explicit expressions for K and Laplace in discrete Fourier space, given the CIC and NGP weight functions and discrete derivative operators used in the code. This allows you to compute the growth rate of each mode individually.

The next step is to do a Taylor series expansion for small k/k_Ny, for which the first term gives the (k/k_Ny)^4 factor - this factor is easy to understand because in the continuum limit K~Laplace, and therefore

K/Laplace = 1 + c(n) (k/k_Ny)^2 + ...

is inevitable. c(n) here is a function of the direction of the k-vector, and due to the symmetries of the grid it can only be a monopole plus an l=4 spherical harmonic. Explicit computation gives

c(n) = -(4/5) pi^5/2 [Y_0^0(n) - (1/3) Y_4^0(n) - sqrt(5/126) Y_4^4(n) - sqrt(5/126) Y_4^-4(n)]

So this already gives you the modified growth as a function of n and k/k_Ny for k/k_Ny << 1. The power spectrum estimator is constructed by integrating over the directions. This gives the numerical constant in my final formula, which I quoted as 6.675 but which more precisely is 12 pi^4 / 175.

> Could you maybe quickly plot this curve to see where this effect becomes relevant?
> The result for different redshifts is attached, I assumed the K_Ny=\pi* N_grid/Boxsize .

This is excellent. It shows that your resolution was good enough that this effect only becomes important right at the Nyqvist wavenumber, and the phi' you get from gevolution should be quite safe from discretization errors (at first order). Had you chosen Ngrid=1024 or even 512, the smallest scales would have been quite contaminated, as the curves would be shifted upwards by a factor of 16 or 256, respectively.

Let me know if you have further questions.

> Maybe we can discuss this in the skype meeting. I think one testbed could be to consider plane symmetric configurations first. For these it turns out that K=Laplacian (as an identity) for any k/k_Ny - in other words, my function c(n) vanishes along the principal axes. These modes will therefore have the correct linear growth despite the discretization.

Its a very good idea. Also it looks straightforward to me. I'll try it.

Best,
Farbod

> On 26 Apr 2018, at 22:28, Julian Adamek <julian.adamek@qmul.ac.uk> wrote:
>
> Dear Farbod,
>
> > Exactly. But unfortunately for my local tests at most I can run with N_grid=256, which the interesting scales are contaminated by the error!
>
> Maybe we can discuss this in the skype meeting. I think one testbed could be to consider plane symmetric configurations first. For these it turns out that K=Laplacian (as an identity) for any k/k_Ny - in other words, my function c(n) vanishes along the principal axes. These modes will therefore have the correct linear growth despite the discretization.
>
> Another advantage would be that you can solve the evolution in Mathematica and compare to your implementation in gevolution.
>
> Best wishes,
> Julian

$$\pi'' + \mathcal{H}(1-3w)\pi' + 3\mathcal{H}\Big(-c_s^2+w\Big)\Psi - \Psi' - 3c_s^2\Phi' + \Big(3\mathcal{H}^2(c_s^2-w)+\mathcal{H}'(1-3c_s^2)\Big)\pi$$
$$- c_s^2\nabla^2\pi - 2c_s^2\Phi\nabla^2\pi + (1-c_s^2)\Psi\nabla^2\pi + 3c_s^2\mathcal{H}(1+w)\pi\nabla^2\pi - (1-c_s^2)(\mathcal{H}\pi+\pi')\nabla^2\pi$$
$$+ c_s^2\nabla\Phi.\nabla\pi - (2c_s^2-1)\nabla\Psi.\nabla\pi + \frac{\mathcal{H}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi.\nabla\pi - 2(1-c_s^2)\nabla\pi.\nabla(\mathcal{H}\pi+\pi') = 0 \tag{1}$$

$$T_0^0(Gev) = \Omega_{kess}^0 a^{-3w}\left[1+\frac{1+w}{c_s^2}\left(-3\mathcal{H}c_s^2\pi - \Psi + (\pi'+\mathcal{H}\pi) - (1-2c_s^2)\frac{(\vec{\nabla}\pi)^2}{2}\right)\right]$$
$$T_0^i(Gev) = -\Omega_{kess}^0 a^{-3w}(1+w)\Big[1-(\frac{1}{c_s^2}-1)\frac{(\vec{\nabla}\pi)^2}{2}\Big]\partial_i\pi$$
$$T_j^i(Gev) = w\,\Omega_{kess}^0 a^{-3w}\left(1+\frac{1+w}{w}\Big[-3\mathcal{H}w\pi - \Psi + (\pi'+\mathcal{H}\pi) - \frac{(\vec{\nabla}\pi)^2}{2}\Big]\delta_j^i + \frac{1+w}{w}\delta^{ik}\partial_k\pi\partial_j\pi\right) \tag{2}$$

we take π and ζ defined as following as new set of variables,

$$\zeta \doteq -\Psi + \pi' + \mathcal{H}\pi, \tag{3}$$

After substitution $\pi' \to \zeta + \Psi - \mathcal{H}\pi$ we get the following expression for stress tensor according to mathematica, which is very clear,

```
= T00 = (1 + w)/cs2 (-3 H cs2 ππ[x, t] - Ψ + πprime + H ππ[x, t] - (1 - 2 cs2) (∂x ππ[x, t] ∂x ππ[x, t]));

= πprime = ζ + Ψ - H ππ[x, t]

= ζ + Ψ - H ππ[x, t]

= T00

= ((1 + w) (ζ - 3 cs2 H ππ[x, t] - (1 - 2 cs2) ππ^(1,0)[x, t]^2))/cs2
```

So the stress tensor reads,

$$T_0^0(Gev) = \Omega^0_{kess} a^{-3w}\left[1+\frac{1+w}{c_s^2}\left(\zeta - 3\mathcal{H}c_s^2\pi - (1-2c_s^2)\frac{(\vec{\nabla}\pi)^2}{2}\right)\right]$$

$$T_0^i(Gev) = -\Omega^0_{kess} a^{-3w}(1+w)\Big[1-(\frac{1}{c_s^2}-1)\frac{(\vec{\nabla}\pi)^2}{2}\Big]\partial_i\pi$$

$$T_j^i(Gev) = w\,\Omega^0_{kess} a^{-3w}\left(1+\frac{1+w}{w}\Big[-3\mathcal{H}w\pi+\zeta-\frac{(\vec{\nabla}\pi)^2}{2}\Big]\delta^i_j+\frac{1+w}{w}\delta^{ik}\partial_k\pi\partial_j\pi\right) \quad (4)$$

After substitution in mathematica we get,

$$\begin{aligned}&\zeta' - (\mathcal{H}\pi)' + \Psi' + \mathcal{H}(1-3w)(\zeta - \mathcal{H}\pi + \Psi) + 3\mathcal{H}\Big(-c_s^2+w\Big)\Psi - \Psi' - 3c_s^2\,\Phi' + \Big(3\mathcal{H}^2(c_s^2-w)+\mathcal{H}'(1-3c_s^2)\Big)\pi\\ &- c_s^2\nabla^2\pi - 2c_s^2\Phi\nabla^2\pi + (1-c_s^2)\Psi\nabla^2\pi + 3c_s^2\mathcal{H}(1+w)\pi\nabla^2\pi - (1-c_s^2)(\zeta+\Psi)\nabla^2\pi\\ &+ c_s^2\nabla\Phi.\nabla\pi - (2c_s^2-1)\nabla\Psi.\nabla\pi + \frac{\mathcal{H}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi.\nabla\pi - 2(1-c_s^2)\nabla\pi.\nabla(\zeta+\Psi) = 0\end{aligned} \quad (5)$$

The linear part simplification gives the same equation we had before and we can rewrite the equation as following,

$$\begin{aligned}&\zeta' - 3w\mathcal{H}\zeta + 3c_s^2\Big(\mathcal{H}^2-\mathcal{H}'\Big)\pi - 3c_s^2\Big(\Phi'+\mathcal{H}\Psi\Big) - c_s^2\nabla^2\pi\\ &- 2c_s^2\Phi\nabla^2\pi + (1-c_s^2)\Psi\nabla^2\pi + 3c_s^2\mathcal{H}(1+w)\pi\nabla^2\pi - (1-c_s^2)(\zeta+\Psi)\nabla^2\pi\\ &+ c_s^2\nabla\Phi.\nabla\pi - (2c_s^2-1)\nabla\Psi.\nabla\pi + \frac{\mathcal{H}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi.\nabla\pi - 2(1-c_s^2)\nabla\pi.\nabla(\zeta+\Psi) = 0\end{aligned} \quad (6)$$

The other equation is like before,

$$\pi' = \zeta + \Psi - \mathcal{H}\pi \quad (7)$$

We actually did nothing except substituting the π' and in non-linear part the substitution is very straightforward. Just note that in Gevolution to compute $\nabla\pi.\nabla(\zeta+\Psi)$ we use the symmetric derivative as follwoing,

$$\nabla\pi.\nabla(\zeta+\Psi) = \frac{1}{4dx^2}\sum_i\Big[\pi(x_i+1)-\pi(x_i-1)\Big]\Big[\big(\zeta(x_i+1)+\Psi(x_i+1)\big)-\big(\zeta(x_i-1)+\Psi(x_i-1)\big)\Big] \quad (8)$$

where "1/4" coefficient appear since we are using symmetric derivative and using points with distance two. "x_i" is the lattice coordinate.

1.1 Solving the equation by leap-frog method

Here we explain how we solve the two equations using leap-frog method,

1.1.1 π equation

for the π equation we have,

$$\pi_{n+1} = \pi_n + \pi'_{n+\frac{1}{2}}\Delta\tau \quad (9)$$

$$\pi'_{n+\frac{1}{2}} = \zeta_{n+\frac{1}{2}} - \mathcal{H}_{n+\frac{1}{2}}\pi_{n+\frac{1}{2}} + \Psi_{n+\frac{1}{2}} \tag{10}$$

According to the definition, since the equation for updating π is linear we do not have any trouble with non-linear terms and this part is done like linear equations.
The background part is better to be updated before this step to have a_{kess} at (n+1/2) and then having $\mathcal{H}(n+1/2)$. Just note that making ζ updating at half steps help us because we need all the terms in $\zeta'(n)$ at integer steps.
Since the scalar field Stress energy tensor must be synchronized with particles stress tensor, we need to have all the variables at the same step which is n, so we need to write all the terms at step $n+\frac{1}{2}$ in terms of the values at step n and $n+1$ as following, of course except ζ which we have it at $n+1/2$. The easiest model to calculate $F_{n+\frac{1}{2}}$ is by taking average of the next and last step, so

$$\pi_{n+\frac{1}{2}} = \frac{\pi_{n+1} + \pi_n}{2} \tag{11}$$

and the same for all the other variables at step $n+\frac{1}{2}$.

For π we have,

$$\pi_{n+1} = \pi_n + \Delta\tau\left[\zeta_{n+\frac{1}{2}} - \mathcal{H}_{n+\frac{1}{2}}(\frac{\pi_{n+1}+\pi_n}{2}) + \Psi_{n+\frac{1}{2}}\right] \tag{12}$$

$$\pi_{n+1} = \frac{1}{1+\mathcal{H}_{n+\frac{1}{2}}\Delta\tau/2}\left[\pi_n + \Delta\tau\left[\zeta_{n+\frac{1}{2}} - \mathcal{H}_{n+\frac{1}{2}}\frac{\pi_n}{2} + \Psi_{n+\frac{1}{2}}\right]\right] \tag{13}$$

As it is clear from the formula we don't have access to the $\Psi_{n+\frac{1}{2}}$ and Ψ_{n+1} , so we use the extrapolation to have them in next half steps,

$$\Psi_{n+\frac{1}{2}} = \Psi_n + \Psi'_n\frac{d\tau}{2} \tag{14}$$

Moreover to have Ψ'_n we use the following formula by saving Ψ at two different steps!

$$\Psi'_n = \frac{\Psi_n - \Psi_{n-1}}{d\tau} \tag{15}$$

1.1.2 ζ equation

We chose to update ζ at half steps since then we need its derivative at integer steps and make the computation simplified. Moreover first π is updated to have it at step $n+1$, then ζ is updated to get the values at $n+3/2$ and from taking the average of the two steps $(n+1/2)$ and $(n+3/2)$ we get ζ at step n which is synchronized with particles stress tensor at step n.

$$\zeta_{n+\frac{3}{2}} = \zeta_{n+\frac{1}{2}} + \zeta'_{n+1}\Delta\tau \tag{16}$$

Just note that in the last step π is updated, so at the current state we have $\zeta_{n+1/2}$, π_{n+1} and ζ_n . $\zeta_{n+1/2}$ which is obtained in the previous loop, ζ_n which in the previous loop was the average of $\zeta_{n-1/2}$ and $\zeta_{n+1/2}$, and π_n was recently updated.
ζ'_n in the equation 26 reads from the differential equation as following,

$$\zeta'_{n+1} = 3\mathcal{H}_{n+1}(w\zeta_{n+1} + c_s^2\Psi_{n+1}) - c_s^2(3\mathcal{H}_{n+1}^2 - 3\mathcal{H}'_{n+1})\pi_{n+1} + 3c_s^2\Phi'_{n+1} + c_s^2\nabla^2\pi_{n+1} \tag{17}$$

$$+ 2c_s^2\Phi_{n+1}\nabla^2\pi_{n+1} - (1-c_s^2)\Psi_{n+1}\nabla^2\pi_{n+1} - 3c_s^2\mathcal{H}_{n+1}(1+w)\pi_{n+1}\nabla^2\pi_{n+1} + (1-c_s^2)(\zeta_{n+1}+\Psi_{n+1})\nabla^2\pi_{n+1}$$
$$- c_s^2\nabla\Phi_{n+1}.\nabla\pi_{n+1} + (2c_s^2-1)\nabla\Psi_{n+1}.\nabla\pi_{n+1} - \frac{\mathcal{H}_{n+1}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi_{n+1}.\nabla\pi_{n+1} + 2(1-c_s^2)\nabla\pi_{n+1}.\nabla(\zeta_{n+1}+\Psi_{n+1}) \tag{18}$$

Since we need to have ζ_n at ζ'_{n+1} so we write $\zeta_{n+1} = \frac{\zeta_{n+3/2}+\zeta_{n+1/2}}{2}$
We get,

$$\begin{aligned}\zeta_{n+\frac{3}{2}} =&\zeta_{n+\frac{1}{2}} + \Delta\tau\Bigg[3\mathcal{H}_{n+1}(w\frac{\zeta_{n+\frac{3}{2}}+\zeta_{n+\frac{1}{2}}}{2} + c_s^2\Psi_{n+1}) - c_s^2(3\mathcal{H}_{n+1}^2 - 3\mathcal{H}'_{n+1})\pi_{n+1} + 3c_s^2\Phi'_{n+1} + c_s^2\nabla^2\pi_{n+1}\\
&+ 2c_s^2\Phi_{n+1}\nabla^2\pi_{n+1} - (1-c_s^2)\Psi_{n+1}\nabla^2\pi_{n+1} - 3c_s^2\mathcal{H}_{n+1}(1+w)\pi_{n+1}\nabla^2\pi_{n+1} + (1-c_s^2)\Big(\frac{\zeta_{n+3/2}+\zeta_{n+1/2}}{2} + \Psi_{n+1}\Big)\nabla^2\pi_{n+1}\\
&- c_s^2\nabla\Phi_{n+1}.\nabla\pi_{n+1} + (2c_s^2-1)\nabla\Psi_{n+1}.\nabla\pi_{n+1} - \frac{\mathcal{H}_{n+1}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi_{n+1}.\nabla\pi_{n+1}\\
&+ 2(1-c_s^2)\nabla\pi_{n+1}.\nabla\Big(\frac{\zeta_{n+3/2}+\zeta_{n+1/2}}{2} + \Psi_{n+1}\Big)\Bigg] \qquad (19)\end{aligned}$$

As one can see the term $\nabla\Big(\frac{\zeta_{n+3/2}+\zeta_{n+1/2}}{2} + \Psi_{n+1}\Big)$ is somehow problematic, since we cannot factorize the $\zeta_{n+3/2}$ and $\zeta_{n+1/2}$ simply. Here we use the predictor-corrector method to solve this equation.
As the first guess we take $\zeta_{n+1} = \zeta_{n+1/2}$, which means we have neglected the term $\zeta'_{n+1/2}d\tau/2$, then we get the following equation,

$$\begin{aligned}\zeta_{n+\frac{3}{2}} =&\zeta_{n+\frac{1}{2}} + \Delta\tau\Bigg[3\mathcal{H}_{n+1}(w\frac{\zeta_{n+\frac{3}{2}}+\zeta_{n+\frac{1}{2}}}{2} + c_s^2\Psi_{n+1}) - c_s^2(3\mathcal{H}_{n+1}^2 - 3\mathcal{H}'_{n+1})\pi_{n+1} + 3c_s^2\Phi'_{n+1} + c_s^2\nabla^2\pi_{n+1}\\
&+ 2c_s^2\Phi_{n+1}\nabla^2\pi_{n+1} - (1-c_s^2)\Psi_{n+1}\nabla^2\pi_{n+1} - 3c_s^2\mathcal{H}_{n+1}(1+w)\pi_{n+1}\nabla^2\pi_{n+1} + (1-c_s^2)\Big(\frac{\zeta_{n+3/2}+\zeta_{n+1/2}}{2} + \Psi_{n+1}\Big)\nabla^2\pi_{n+1}\\
&- c_s^2\nabla\Phi_{n+1}.\nabla\pi_{n+1} + (2c_s^2-1)\nabla\Psi_{n+1}.\nabla\pi_{n+1} - \frac{\mathcal{H}_{n+1}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi_{n+1}.\nabla\pi_{n+1}\\
&+ 2(1-c_s^2)\nabla\pi_{n+1}.\nabla\Big(\zeta_{n+1/2} + \Psi_{n+1}\Big)\Bigg] \qquad (20)\end{aligned}$$

Factorizing $\zeta_{n+3/2}$ gives,

$$\begin{aligned}\zeta_{n+\frac{3}{2}}\Big[1 - 3\mathcal{H}_{n+1}w\Delta\tau/2 - (1-c_s^2)\nabla^2\pi_{n+1}\Delta\tau/2\Big] =& \zeta_{n+\frac{1}{2}} + \Delta\tau\Bigg[3\mathcal{H}_{n+1}(w\frac{\zeta_{n+\frac{1}{2}}}{2} + c_s^2\Psi_{n+1}) - c_s^2(3\mathcal{H}_{n+1}^2 - 3\mathcal{H}'_{n+1})\pi_{n+1} + 3c_s^2\Phi'_{n+1}\\
&+ c_s^2\nabla^2\pi_{n+1} + 2c_s^2\Phi_{n+1}\nabla^2\pi_{n+1} - (1-c_s^2)\Psi_{n+1}\nabla^2\pi_{n+1} - 3c_s^2\mathcal{H}_{n+1}(1+w)\pi_{n+1}\nabla^2\pi_{n+1} + (1-c_s^2)\Big(\frac{\zeta_{n+1/2}}{2} + \Psi_{n+1}\Big)\nabla^2\pi_{n+1}\\
&- c_s^2\nabla\Phi_{n+1}.\nabla\pi_{n+1} + (2c_s^2-1)\nabla\Psi_{n+1}.\nabla\pi_{n+1} - \frac{\mathcal{H}_{n+1}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi_{n+1}.\nabla\pi_{n+1}\\
&+ 2(1-c_s^2)\nabla\pi_{n+1}.\nabla\Big(\zeta_{n+1/2} + \Psi_{n+1}\Big)\Bigg] \qquad (21)\end{aligned}$$

Finally we can write the below equation,

$$\zeta_{n+\frac{3}{2}} = \frac{1}{1-3\mathcal{H}_{n+1}w\Delta\tau/2-(1-c_s^2)\nabla^2\pi_{n+1}\Delta\tau/2}\Bigg[\zeta_{n+\frac{1}{2}}+\Delta\tau\Bigg(3\mathcal{H}_{n+1}(w\frac{\zeta_{n+\frac{1}{2}}}{2}+c_s^2\Psi_{n+1})-c_s^2(3\mathcal{H}_{n+1}^2-3\mathcal{H}'_{n+1})\pi_{n+1}+3c_s^2\Phi'_{n+1}$$
$$+c_s^2\nabla^2\pi_{n+1}+2c_s^2\Phi_{n+1}\nabla^2\pi_{n+1}-(1-c_s^2)\Psi_{n+1}\nabla^2\pi_{n+1}-3c_s^2\mathcal{H}_{n+1}(1+w)\pi_{n+1}\nabla^2\pi_{n+1}+(1-c_s^2)\Big(\frac{\zeta_{n+1/2}}{2}+\Psi_{n+1}\Big)\nabla^2\pi_{n+1}$$
$$-c_s^2\nabla\Phi_{n+1}.\nabla\pi_{n+1}+(2c_s^2-1)\nabla\Psi_{n+1}.\nabla\pi_{n+1}-\frac{\mathcal{H}_{n+1}}{2}\Big(2+3w+c_s^2\Big)\nabla\pi_{n+1}.\nabla\pi_{n+1}$$
$$+2(1-c_s^2)\nabla\pi_{n+1}.\nabla\big(\zeta_{n+1/2}+\Psi_{n+1}\big)\Bigg)\Bigg] \tag{22}$$

which at linear order we get the same equation as we had before,

$$\zeta_{n+\frac{3}{2}} = \frac{1}{1-3\mathcal{H}_{n+1}w\Delta\tau/2}\Bigg[\zeta_{n+\frac{1}{2}}+\Delta\tau\Bigg(3\mathcal{H}_{n+1}\Big(\frac{w\zeta_{n+\frac{1}{2}}}{2}+c_s^2\Psi_{n+1}\Big)-c_s^2(3\mathcal{H}_{n+1}^2-3\mathcal{H}'_{n+1})\pi_{n+1}+3c_s^2\Phi'_{n+1}+c_s^2\nabla^2\pi_{n+1}\Bigg)\Bigg] \tag{23}$$

It is very important to check everything for the first loop specifically! For the first loop we first compute $\zeta^{(-1/2)}$ by $\zeta_{-1/2}=\zeta_0-\zeta'_0 d\tau/2$ and then ζ is updated to have $\zeta^{(1/2)}$ from the values of $\Psi^{(0)}$ but we take $\Phi'^{(0)}=0$ which is an approximation in our scheme.
Then we go through the procedure which is the same for all loops (first loop included), which we update π to have it at step 1 and then we update ζ to have it at step 3/2 during all of these procedures, Φ is assumed to be constant and we use $\Phi^{(1/2)}=\Phi^{(0)}$ since in the first loop Φ' is zero! So this is an approximation that in the first loop we take $\Psi'=0$ and use the same Φ at 1/2 and 0 step the same. But in the other loops everything seems correct as following,
To compute $\zeta_{-1/2}$ we just use the linear equation for ζ'_0 and also neglect Φ'_0.
The whole updating loop is as following,
NOTE!
Another way to update the fields is like velocities and positions and to compute $\zeta(n+1/2)$ just from $\zeta(n-1/2)$ and then compute $\zeta'(n)$, $\zeta'(n+1/2)$ and then extrapolate to compute $\zeta(n+1)$.

Approximations:

The important approximations here are:
1- At the first loop we take $\Phi'=0$
2- To compute $\zeta_{-1/2}$ we neglect non-linear terms, but it should not be important, since we use predictor corrector method in other loops, which correct the values. 3- The value of $\Phi^{(n+1)}=\Phi^{(n)}+\Phi'^{(n)}\times d\tau$ to compute $\zeta^{n+3/2}$ which again at the first loop we assume $\Phi^{(1)}=\Phi^{(0)}$
4- Since we do not have $\Phi'^{(n+1)}$ we assume that it does not change fast, so we take $\Phi'^{(n+1)}=\Phi'^{(n)}$ or equivalently $\Phi''^{(n)}\approx 0$
All the top approximations can be suppressed by increasing the number of kessence update or decreasing the time stepping of Gevolution.
5- We use predictor-corrector method for couple of times in each loop to correct our bad guess of non-linear term $\nabla\zeta_{n+1}$)

Before talking about predictor-corrector method, we mention the important parts in the Gevolution which is changed for non-linear implementation,

The parameters are the same as linear part, so we just need to check the updatings in the main loop which is as following and the same as linear updating!

```
//**********************
//Kessence - LeapFrog:START
//**********************
  double a_kess=a;
        for (i=0;i<sim.nKe_numsteps;i++)
        {
    //First we update zeta to have it at 0-1/2 just in the first loop
    if(cycle==0)
    {
      for (x.first(); x.test(); x.next())
      {
        //computing zeta(-1/2)
        zeta_half(x) =zeta_integer(x) - 0.5 * dtau * ( 3. * Hconf(a_kess, fourpiG, cosmo) * ( cosmo.
            w_kessence * zeta_integer(x) + cosmo.cs2_kessence * phi(x) - chi(x) ) - cosmo.cs2_kessence *
            (3. * Hconf(a_kess, fourpiG, cosmo) * Hconf(a_kess, fourpiG, cosmo) - 3. * Hconf_prime(a_kess,
             fourpiG, cosmo) ) * pi_k(x));
        //Approximations: 1-The linear definition of derivative
        //                2-phi_prime = 0
        //                3- cs^2 Laplace pi =0
        //                4- neglecting the non-linear terms for computing zeta(-1/2)
        //Phi_prime is omitted since in the first loop is zero
        // We also have neglected Laplace term since Laplace itself is small and is multiplied to cs^2
            which is very suppressed!
        // TO MAKE SURE THE APPROXIMATIONS WORK WELL, WE NEED TO INCREASE THE PRECISION AND SEE THE
            IMPROVEMENTS!
      }
      //Updating zeta to get zeta(1/2) and zeta(0) just in the first loop
      // In sum: zeta(1/2) = zeta(-1/2)=zeta(0) + zeta'(0) dtau for the first loop
      update_zeta(dtau/ sim.nKe_numsteps, dx, a_kess, phi, phi_old, chi, chi_old, pi_k, zeta_half,
          zeta_integer, cosmo.Omega_kessence, cosmo.w_kessence, cosmo.cs2_kessence, Hconf(a_kess, fourpiG,
           cosmo), Hconf_prime(a_kess, fourpiG, cosmo));
      zeta_half.updateHalo();
      zeta_integer.updateHalo();
    }

    //Since we have pi(n+1)=pi(n) + pi'(n+1/2), and in pi'(n+1/2) we have H(n+1/2) we update the
        background before updating the pi to have H(n+1/2)
    rungekutta4bg(a_kess, fourpiG, cosmo,  dtau  / sim.nKe_numsteps / 2.0);
    //First we update pi to have it at n+1 (at first loop from the value at (0) and the value of zeta at
        1/2 and H(n+1/2) we update pi at (1))
    update_pi_k(dtau/ sim.nKe_numsteps, dx, a_kess, phi, phi_old, chi, chi_old, pi_k, zeta_half,
        zeta_integer, cosmo.Omega_kessence, cosmo.w_kessence, cosmo.cs2_kessence, Hconf(a_kess, fourpiG,
        cosmo),Hconf_prime(a_kess, fourpiG, cosmo)); // H_old is updated here in the function
              pi_k.updateHalo();
    // Now we have pi(n+1) and a_kess(n+1/2) so we update background by halfstep to have a_kess(n+1) to
        calculate zeta'(n+1) to have zeta(n+1/2)=zeta(n-1/2) + zeta'(n+1) dtau
    rungekutta4bg(a_kess, fourpiG, cosmo,  dtau  / sim.nKe_numsteps / 2.0 );
    //Now from the values of zeta at step (1/2) we calculate zeta(3/2) and then we calculate zeta(1) which
        is synched with pi(1)
    update_zeta(dtau/ sim.nKe_numsteps, dx, a_kess, phi, phi_old, chi, chi_old, pi_k, zeta_half,
        zeta_integer, cosmo.Omega_kessence, cosmo.w_kessence, cosmo.cs2_kessence, Hconf(a_kess, fourpiG,
        cosmo), Hconf_prime(a_kess, fourpiG, cosmo));
    zeta_half.updateHalo();
    zeta_integer.updateHalo();

        }
#ifdef BENCHMARK
    kessence_update_time += MPI_Wtime() - ref_time;
    ref_time = MPI_Wtime();
#endif
//**********************
//Kessence - LeapFrog: End
//**********************
```

The most important changes compared to linear equations occur in the Gevolution.hpp, since we need to add, non-linear terms in the stress tensor correctly, adding non-linear terms in the updating equations and also predictor-corrector method should be added.
The stress tensor of kessence is modified as following,

```
  ////////////////////////
template <class FieldType>
void projection_Tmunu_kessence( Field<FieldType> & T00, Field<FieldType> & T0i, Field<FieldType> & Tij,
   double dx, double a, Field<FieldType> & phi,Field<FieldType> & phi_old, Field<FieldType> & chi, Field<
        FieldType> & pi_k, Field<FieldType> & zeta_integer, double Omega_fld , double w, double cs2, double
        Hcon, double fourpig, int method )
{
    Site xField(phi.lattice());
    double coeff1, coeff2, coeff3, Hdot, psi;
    double gradientpi_squared, Dx_pi_Dx_pi, Dx_pi_Dy_pi, Dx_pi_Dz_pi, Dy_pi_Dy_pi, Dy_pi_Dz_pi,
        Dz_pi_Dz_pi;
    Site x(phi.lattice());
    double gradient_pi2;
    coeff1=Omega_fld*pow(a,-3.*w)*(1.+w)/(cs2);
    coeff2=Omega_fld*pow(a,-3.*w)*(1.+w);

    for (xField.first(); xField.test(); xField.next())
      {
        //***************
        //(D_i pi)^2
        //***************
```

```
      gradientpi_squared =0.25*(pi_k(xField+0) - pi_k(xField-0))* (pi_k(xField+0) - pi_k(xField-0))/(dx*
          dx);
      gradientpi_squared+=0.25*(pi_k(xField+1) - pi_k(xField-1))* (pi_k(xField+1) - pi_k(xField-1))/(dx*
          dx);
      gradientpi_squared+=0.25*(pi_k(xField+1) - pi_k(xField-2))* (pi_k(xField+1) - pi_k(xField-2))/(dx*
          dx);
      //***************
      //(X,X)::::::: > Dx_pi_Dx_pi = GradX(pi).GradX(pi)
      //***************
      Dx_pi_Dx_pi =0.25*(pi_k(xField+0) - pi_k(xField-0))* (pi_k(xField+0) - pi_k(xField-0))/(dx*dx);
      //***************
      //(X,Y)::::::: > Dx_pi_Dy_pi = GradX(pi).GradY(pi) = GradY(pi).GradX(pi)
      //***************
      Dx_pi_Dy_pi =0.25*(pi_k(xField+0) - pi_k(xField-0))* (pi_k(xField+1) - pi_k(xField-1))/(dx*dx);
      //***************
      //(X,Z)::::::: > Dx_pi_Dz_pi = GradX(pi).GradZ(pi)
      //***************
      Dx_pi_Dz_pi =0.25*(pi_k(xField+0) - pi_k(xField-0))* (pi_k(xField+2) - pi_k(xField-2))/(dx*dx);
      //***************
      //(Y,Y)::::::: > Dy_pi_Dy_pi = GradY(pi).GradY(pi)
      //***************
      Dy_pi_Dy_pi =0.25*(pi_k(xField+1) - pi_k(xField-1))* (pi_k(xField+1) - pi_k(xField-1))/(dx*dx);
      //***************
      //(Y,Z)::::::: > Dy_pi_Dz_pi = Grady(pi).Gradz(pi)
      //***************
      Dy_pi_Dz_pi =0.25*(pi_k(xField+1) - pi_k(xField-1))* (pi_k(xField+2) - pi_k(xField-2))/(dx*dx);
      //***************
      //(Z,Z)::::::: > Dz_pi_Dz_pi = Gradz(pi).Gradz(pi)
      //***************
      Dz_pi_Dz_pi =0.25*(pi_k(xField+2) - pi_k(xField-2))* (pi_k(xField+2) - pi_k(xField-2))/(dx*dx);
      //***************
      psi= phi(xField) - chi(xField);

      //************************
      //STRESS TENSOR COMPONENTS
      //************************
      // 0-0-component: (Time,Time)
      T00(xField)       = - coeff1 * ( -3. * cs2 * Hcon * pi_k(xField) + zeta_integer(xField)
                          /*Non-linear*/ - (1. - 2. * cs2) * gradientpi_squared / 2.  );
      //*****************************************************************************
      // 1-1-component: (X,X)
      Tij(xField, 0, 0) = + coeff2 * (-3.* w * Hcon* pi_k(xField) + zeta_integer(xField)
                          /*Non-linear*/ - gradientpi_squared / 2.  + Dx_pi_Dx_pi );
      //*****************************************************************************
      // 2-2-component: (Y,Y)
      Tij(xField, 1, 1) = + coeff2 * (-3.* w * Hcon* pi_k(xField) +   zeta_integer(xField)
                          /*Non-linear*/ - gradientpi_squared / 2.  + Dy_pi_Dy_pi );
      //*****************************************************************************
      // 3-3-component: (Z,Z)
      Tij(xField, 2, 2) = + coeff2 * (-3.* w * Hcon* pi_k(xField) +   zeta_integer(xField)
                          /*Non-linear*/ - gradientpi_squared / 2.  + Dz_pi_Dz_pi );
      //*****************************************************************************
      // 1-2-component: (X,Y)
      Tij(xField, 0, 1) = + coeff2 * ( /*Non-linear*/ Dx_pi_Dy_pi);
      //*****************************************************************************
      // 1-3-component: (X,Z)
      Tij(xField, 0, 2) = + coeff2 * (/*Non-linear*/ Dx_pi_Dz_pi);
      //*****************************************************************************
      // 2-3-component: (Y,Z)
      Tij(xField, 1, 2) = + coeff2 * (/*Non-linear*/ Dy_pi_Dz_pi);
      //*****************************************************************************

      //*****************************
      //In the case of Vector parabolic
      //*****************************
      if(method==1) // method=1 Turn on vector elliptic
      {
                                    // T01:(Time,X)
        T0i(xField, 0)  =  -coeff2 * (1. - /*Non-linear*/ (1./cs2 -1.) * gradientpi_squared / 2.) *
                  (pi_k(xField + 0) - pi_k(xField - 0)) / (2. * dx);
        //*****************************************************************************
                                    // T02:(Time,Y)
        T0i(xField, 1)  =  -coeff2 *  (1. - /*Non-linear*/ (1./cs2 -1.) * gradientpi_squared / 2.) *
                  (pi_k(xField + 1) - pi_k(xField - 1)) / (2. * dx);
        //*****************************************************************************
        // T03:(Time,Z)
        T0i(xField, 2)  =  -coeff2 *  (1. - /*Non-linear*/ (1./cs2 -1.) * gradientpi_squared / 2.) *
                  (pi_k(xField + 2) - pi_k(xField - 2)) / (2. * dx);
        //*****************************************************************************
      }
    }
  }
```

The updating π equation does not change, since by definition is linear,

```
      void update_pi_k( double dtau, double dx,double a, Field<FieldType> & phi, Field<FieldType> &
          phi_old, Field<FieldType> & chi,Field<FieldType> & chi_old, Field<FieldType> & pi_k, Field<
          FieldType> &,  Field<FieldType> & zeta_half , double Omega_fld ,double w, double cs2, double
          Hcon, double  H_prime)
                    {
      double psi, psi_prime, psi_half;
      double Coeff1 = 1./(1. + Hcon * dtau/2.);
                      Site x(phi.lattice());
                      for (x.first(); x.test(); x.next())
                        {
          psi=phi(x) - chi(x);
          psi_prime= ((phi(x) - chi(x))-(phi_old(x) - chi_old(x)))/dtau;
          psi_half= psi + psi_prime * dtau/2.; //psi_half (n+1/2) = psi(n) + psi_prime'(n) dtau/2
```

```
        //*************************************
        //pi Updating which is linear by definition
        //*************************************
        pi_k(x)=Coeff1 * (pi_k(x)  + dtau * ( zeta_half(x) - Hcon * pi_k(x)/2. + psi_half ) ); //
            pi_k(n+1)
        //*****************************************************************************
                        }
                    }
```

The most challenging part is implementing non-linear terms in the updating ζ equation and since we use some approximations we first introduce the predictor-corrector idea and then put the C++ code.

1.1.2.1 Predictor-corrector method

This is an important part which should be done to have the errors propagation under the control! As we have explained already in the first loop of updating we take $\nabla\zeta_{n+1} = \nabla\zeta_{n+1/2}$ which is just prediction and then we calculate $\zeta_{n+3/2}$ from the predictor and then we compute ζ_{n+1} by averaging the two ζ at half steps.
So at the end we have ζ_{n+1} which we did not have in the start, so we are going to use ζ_{n+1} as new value to compute $\zeta_{n+3/2}$ and then again we calculate new corrected ζ_{n+1} and we take it as initial value to compute the corrected $\zeta_{n+3/2}$ and ζ_{n+1}
The best way to do it is first to check that ζ_n from the next order corrector is near to original one (like the relative error $2\frac{\zeta_1-\zeta_2}{\zeta_1+\zeta_2} < 0.01$, which gives less than 1% error).
The code for predictor corrector method in C++ is written in below, note that all the lines have comments make the code easier to follow,

```
// We use predictor corrector method to calculate \zeta precisely for non-linear case. (for linear
      equation is does not make better)
                template <class FieldType>
                void update_zeta(double dtau, double dx,double a, Field<FieldType> & phi, Field<
                    FieldType> & phi_old, Field<FieldType> & chi,Field<FieldType> & chi_old, Field
                    <FieldType> & pi_k, Field<FieldType> & zeta_half , Field<FieldType> &
                    zeta_integer, double Omega_fld ,double w, double cs2, double Hcon, double
                    H_prime )
                {
double Gradphi_Gradpi, Gradpsi_Gradpi, Gradpi_Gradpi, GradPsiZeta_Gradpi, Dx_psi, Dy_psi, Dz_psi;
                  double C1, C2, C3, psi, psi_old, psi_prime, phi_prime, Laplacian_pi,
                      zeta_old_half;
//Since a_kess is at (n+1) so H_prime is at (n+1) which is needed to calculate zeta(n+1/2)
//*************************************************************
//Coefficient two, H(n+1), H_prime(n+1) since BG already updated
//*************************************************************
C2 = cs2 * (3. * Hcon * Hcon - 3. * H_prime );
//*************************************************************
//Coefficient three, H(n+1), H_prime(n+1) since BG already updated
//*************************************************************
C3 = (2. + 3. * w + cs2 ) *  Hcon/2.;

                        Site x(phi.lattice());
                        for (x.first(); x.test(); x.next())
                                {
    //*************************************************************
    //Laplace pi, pi(n+1) since pi updated before zeta and a_kess(n+1)
    //*************************************************************
                                    Laplacian_pi= pi_k(x-0) + pi_k(x+0) - 2. * pi_k(x);
                                    Laplacian_pi+=pi_k(x+1) + pi_k(x-1) - 2. * pi_k(x);
                                    Laplacian_pi+=pi_k(x+2) + pi_k(x-2) - 2. * pi_k(x);
    Laplacian_pi= Laplacian_pi/(dx*dx);
    //*****************************************
    //psi(n), APPROXMIATION:we take psi(n+1)=psi(n)
    //*****************************************
                                    psi=phi(x) - chi(x);
    //**************************
    //Coefficient one, H( at n+1)
    //**************************
    C1 = 1./(1. -  3. * Hcon * w  * dtau/2. - (1. - cs2) *  Laplacian_pi * dtau/2.);
    //*****************************************
    //phi'(n), APPROXMIATION:we take phi'(n+1)=phi'(n)
    //*****************************************
                                phi_prime= (phi(x) - phi_old(x))/dtau; //phi_prime(n+1) since we
                                    want to use it to compute zeta (n+3/2)
    //NOTE: We dont have phi'(n+1) since it will be updated by particles later, but we  remains
        contant and phi_prime(n) = phi_prime(n+1) or take the second derivative for this period
        approximately to zero!
    //*******************************
    //psi'(n)
    //psi(n+1) = psi(n) + psi'(n) dtau
    //*******************************
    psi_prime= ((phi(x) - chi(x))-(phi_old(x) - chi_old(x)))/dtau;
```

```
psi = psi + psi_prime * dtau;
//**********
//Grad_i Psi
//**********
Dx_psi = ((phi(x + 0) - chi(x + 0)) - (phi(x - 0) - chi(x - 0)));
Dy_psi = ((phi(x + 1) - chi(x + 1)) - (phi(x - 1) - chi(x - 1)));
Dz_psi = ((phi(x + 2) - chi(x + 2)) - (phi(x - 2) - chi(x - 2)));
//******************
//Grad_phi . Grad_pi
//******************
Gradphi_Gradpi= 0.25 * (phi(x + 0)  - phi(x - 0)) * (pi_k(x + 0) - pi_k(x - 0)) / (dx * dx);
Gradphi_Gradpi+=0.25 * (phi(x + 1)  - phi(x - 1)) * (pi_k(x + 1) - pi_k(x - 1)) / (dx * dx);
Gradphi_Gradpi+=0.25 * (phi(x + 2)  - phi(x - 2)) * (pi_k(x + 2) - pi_k(x - 2)) / (dx * dx);
//******************
//Grad_psi . Grad_pi
//******************
Gradpsi_Gradpi= 0.25 * (Dx_psi) * (pi_k(x+0) - pi_k(x-0)) / (dx * dx);
              Gradpsi_Gradpi+=0.25 * (Dy_psi) * (pi_k(x+1) - pi_k(x-1))/ (dx * dx);
              Gradpsi_Gradpi+=0.25 * (Dz_psi) * (pi_k(x+2) - pi_k(x-2))/ (dx * dx);
//*************
//Gradpi_Gradpi
//*************
Gradpi_Gradpi= 0.25 * (pi_k(x + 0)  - pi_k(x - 0)) * (pi_k(x + 0) - pi_k(x - 0)) / (dx * dx);
Gradpi_Gradpi+=0.25 * (pi_k(x + 1)  - pi_k(x - 1)) * (pi_k(x + 1) - pi_k(x - 1)) / (dx * dx);
Gradpi_Gradpi+=0.25 * (pi_k(x + 2)  - pi_k(x - 2)) * (pi_k(x + 2) - pi_k(x - 2)) / (dx * dx);
//*********************************************************
//GradPsiZeta_Gradpi = Grad_pi . Grad_ (zeta + psi)
//Grad_pi . Grad_ (zeta + psi) = Grad_pi (Grad_zeta + Grad_Psi)
//*********************************************************
GradPsiZeta_Gradpi= 0.25* (zeta_half(x+0) - zeta_half(x-0) + Dx_psi) * (pi_k(x+1) - pi_k(x-1))
     / (dx * dx);
GradPsiZeta_Gradpi+=0.25* (zeta_half(x+1) - zeta_half(x-1) + Dy_psi) * (pi_k(x+1) - pi_k(x-1))
     / (dx * dx);
GradPsiZeta_Gradpi+=0.25* (zeta_half(x+2) - zeta_half(x-2) + Dz_psi) * (pi_k(x+2) - pi_k(x-2))
     / (dx * dx);
//***********************
// FULL Updating equation
//***********************
zeta_old_half=zeta_half(x); // zeta(n+1/2)

//************************************
// zeta(n+3/2) = zeta(n+1/2) + zeta'(n+1)
//************************************
zeta_half(x) =        C1 * ( zeta_half(x) + dtau * (
/*Linear(1,2,3)*/       + 3. * Hcon * ( w * zeta_half(x)/2. + cs2 * psi ) - C2 * pi_k(x)
/*Linear(4,5)*/         + 3. * cs2 * phi_prime + cs2 * Laplacian_pi
/*Non-linear(1,2)*/     + 2. * cs2 * phi(x) * Laplacian_pi - (1. - cs2) * psi * Laplacian_pi
/*Non-linear(3)  */     - 3. * cs2 * Hcon * (1. + w) * pi_k(x) * Laplacian_pi
/*Non-linear(4)  */     +(1. - cs2) * (zeta_half(x)/2. + psi) * Laplacian_pi
/*Non-linear(5,6,7)*/   - cs2 * Gradphi_Gradpi + (2. * cs2 -1.) * Gradpsi_Gradpi - C3 *
     Gradpi_Gradpi
/*Non-linear(8)  */     + 2. * (1. - cs2) * GradPsiZeta_Gradpi
                                                    )
                            );
//*********************************************************************
//computing zeta (n+1) by taking average ove zeta(n+3/2) and zeta(n+1/2)
//*********************************************************************
zeta_integer(x)= (zeta_half(x) + zeta_old_half)/2.; //zeta(n+1)

//*********************************************************************
//*********************************************************************
//*********************************************************************
//PREDICTOR-CORRECTOR METHOD
//*********************************************************************
//*********************************************************************
//*********************************************************************

//****************************
//Predictor-corrector variables
//****************************
double zeta_predictor_int_n0, zeta_prime_int_n0, zeta_predictor_half_n1 ;
int n_correcor_steps=10, numerator;
numerator=0;
//****************************************
//n_correcor_steps loops over corrector method
//****************************************
for (int i=1; i<n_correcor_steps+1; i++)
{
//****************************************
//Initiation of the method from the last loop
//****************************************
zeta_predictor_int_n0 = zeta_integer(x);  //zeta (n+1) = (zeta(n+3/2)+zeta(n+1/2)) /2
//*********************************************************
//Corrected: GradPsiZeta_Gradpi = Grad_pi . Grad_ (zeta + psi)
//Grad_pi . Grad_ (zeta + psi) = Grad_pi (Grad_zeta + Grad_Psi)
//Where zeta_integer is the corrected one at step (n+1)
//Before we have zeta_half because of approximation
//*********************************************************
GradPsiZeta_Gradpi= 0.25* (zeta_integer(x+0) - zeta_integer(x-0) + Dx_psi) * (pi_k(x+1) - pi_k
     (x-1)) / (dx * dx);
GradPsiZeta_Gradpi+=0.25* (zeta_integer(x+1) - zeta_integer(x-1) + Dy_psi) * (pi_k(x+1) - pi_k
     (x-1)) / (dx * dx);
GradPsiZeta_Gradpi+=0.25* (zeta_integer(x+2) - zeta_integer(x-2) + Dz_psi) * (pi_k(x+2) - pi_k
     (x-2)) / (dx * dx);
//************************************
// zeta'(n+1) from the new values at n+1
// like zeta_predictor_int_n0 (n+1)
//************************************
zeta_prime_int_n0 =
```

```
        /*Linear(1,2,3)*/       + 3. * Hcon * ( w * zeta_predictor_int_n0 + cs2 * psi ) - C2 * pi_k(x)
        /*Linear(4,5)*/         + 3. * cs2 * phi_prime + cs2 * Laplacian_pi
        /*Non-linear(1,2)*/     + 2. * cs2 * phi(x) * Laplacian_pi - (1. - cs2) * psi * Laplacian_pi
        /*Non-linear(3)  */     - 3. * cs2 * Hcon * (1. + w) * pi_k(x) * Laplacian_pi
        /*Non-linear(4)  */     +(1. - cs2) * (zeta_predictor_int_n0 + psi) * Laplacian_pi
        /*Non-linear(5,6,7)*/   - cs2 * Gradphi_Gradpi + (2. * cs2 -1.) * Gradpsi_Gradpi - C3 *
            Gradpi_Gradpi
        /*Non-linear(8)  */     + 2.* (1. - cs2) * GradPsiZeta_Gradpi;
        //*************************************************************************
        //Computing the zeta(n+3/2) from the new value of zeta'(n+1) and zeta(n+1/2)
        //*************************************************************************
        zeta_predictor_half_n1 = zeta_old_half + zeta_prime_int_n0 * dtau;
        //*****************************************************************************
        //Computing the corredcted zeta(n+1) from the new value of zeta(n+3/2) and zeta(n+1/2)
        //*****************************************************************************
        zeta_predictor_int_n0  = (zeta_predictor_half_n1 + zeta_old_half)/2.;
        //*************************************************************************
        // Now we must check the relative error between the two corrected and predicted ones
            zeta_integer(x)=zeta_predictor_int_n0 and the previous step: zeta_half(x), zeta_integer(x)
        //If the error has reached less than 10^-6% it breacks the loop
        //*************************************************************************************
        if (2.* abs(zeta_integer(x)-zeta_predictor_int_n0)/(zeta_integer(x)+zeta_predictor_int_n0)<1.e
            -8)break;
         zeta_half(x)=zeta_predictor_half_n1; // zeta(n+3/2) after correction
         zeta_integer(x)= (zeta_half(x) + zeta_old_half)/2.; //zeta(n+1) after correction
         numerator++;
        }
        if (numerator==n_correcor_steps) cout << "\033[1;31mbold WARNING: PRECISION ERROR ON KESSENCE
            FIELD ZETA, More than 1% Error\033[0m\n" << '\n';
    }
}
```

As we know although the implementation is checked well but it is very important to design some tests to check the terms as precise as possible.

1.2 Some important changes in the code and the way we solve equations This is completely wrong!

So after skyping with Martin and Julian, we have realized that it is better to update kessence field after particles to have potential at step $n+1$, so if we do this the only change is the fact that Φ_{old} is really $\Phi(n)$ and new Φ is at step (n+1), so to compute $\Phi'(n+1)$. In order to compute $\Phi_{n+1/2}$ which appears in π updating we have,

$$\Psi_{n+1/2} = \frac{\Psi_{n+1} + \Psi_n}{2} \tag{24}$$

where Ψ_{n+1} and Ψ_{n+1} the new and old value of Ψ and to compute Φ_{n+1} and Φ'_{n+1} is very simple! Since we already have Φ_{n+1} because of particles update and to compute Φ'_{n+1} we have,

$$\Phi'_{n+1} = \frac{\Phi_{n+1} - \Phi_n}{d\tau} \tag{25}$$

So we have a small change in the way we calculate the potentials and their derivatives in Gevolution.hpp and the place where we update the kessence fields in the main.hpp Moreover we have adde a integer in the Gevolution, NL_kessence which can get 0 or 1 and in the 0 case we get linear kessence evolution and in case 1 we inclue non-linearity as well. This is wrong! because in any case if we put kessence update after or before particles we dont have Φ at different steps, the potentials are updated in the next loop!
Another discussion I had with Julian on this is,

> Moreover I've realized that our reasoning on updating kessence field after particles updating was wrong. Because the "potentials" are updated in the beginning of the main loop and always particles and kessence fields are one step ahead of them. So in no situation (except saving three potential and extrapolating) we can calculate \Phi' at step (n+1) from Phi(n) and Phi(n-1).

I still think that you never need \Phi' at (n+1). Say gevolution has just computed \Phi at step n (and maybe you still have \Phi at n-1 because you want to compute time derivatives). Next any quantities that live at half-integer time steps (e.g. particle velocities) will be moved from (n-1/2) to (n+1/2) - this update will only depend on quantities that were computed at step n and step (n-1/2). Then all quantities that live at integer time steps (such as particle positions) will be moved from n to n+1 - again this update will only depend on quantities that were computed already, in particular those at (n+1/2) and n. The worst thing that could happen therefore is that you need \Phi' at (n+1/2) which would be annoying because you'd maybe want to compute it using \Phi at (n+1). Is this your problem?

No this is not what we are doing, we are updating the velocities half step ahead of the positions not behind!

1.3 Important changes

Most of the steps in the precious sections are changes because of better ideas!
-Basically we do not need predictor-corrector method, since we update $\zeta(n+1/2)$ from $\zeta(n-1/2)$ using $\zeta'(n)$ which is just a function of $\nabla(\zeta(n))$ which we have it from ζ_{int}
-Now we update both ζ_{int} and ζ_{half} separately, one in zetaupdate the other one in the pi-update, since the backgrounds are in different time! After having chat with Julian as following,

Dear Farbod,

thanks for sending your notes.

I don't understand your reasoning in section 1.1.2. Why do you want to update zeta from (n+1/2) to (n+3/2) before having moved all quantities that are at integer time steps from n to (n+1)? This makes no sense at all. This update should only be done *after* all fields at integer time steps have been moved to (n+1), including of course the gravitational potentials.

It looks to me that all the problems are solved if you do the zeta update (which is at half integer time steps) *before* the pi update (which is at integer time steps). This is how it should be done.

Best wishes,
Julian

Dear Julian,

Thanks for the comments,

> I don't understand your reasoning in section 1.1.2. Why do you want to update zeta from (n+1/2) to (n+3/2) before having moved all quantities that are at integer time steps from n to (n+1)?

I'm not doing it "before". I'm moving zeta from (n+1/2) to (n+3/2) "after" updating pi from n to n+1 but I'm using potentials at step n (I think you mean potentials specifically?)

> It looks to me that all the problems are solved if you do the zeta update (which is at half integer time steps) *before* the pi update (which is at integer time steps). This is how it should be done.

I initially wrote the code to follow exactly this way. But how we compute zeta at n+1 in this way? You think like zeta(n+1) = zeta(n+1/2) + zeta'(n) dtau/2 ?

Dear Farbod,

all I'm saying is that a leapfrog works like this:

n-1/2 -> n+1/2
n -> n+1
n+1/2 -> n+3/2
n+1 -> n+2
.
.
.

You are proposing to do n+1/2 -> n+3/2 *before* n -> n+1 is complete.

We see that it is much better to go over loop in a way that we do not need $\Phi(n+1)$ or $\Phi'(n+1)$ which will be calculated at next loop!
To do so I will change the updatings as following, but most of the main concepts of course remain the same, just the first loop and .. would change. So all the updating equation for ζ and π are of course the same but the way we will go through Leap0frog should be changes as following!

- First we provide the initial condition for $\zeta(0)$, $\pi(0)$, $\Psi(0)$
- Then we update ζ by minus half step from the values at step 0 to get $\zeta(-1/2)$, just at the first loop!
- So we are ready to go over the general loop, we first update ζ to have the value at $\zeta(1/2)$ from $\zeta(-1/2)$ and $\Phi(0)$, $\pi(0)$, we compute $\zeta(1) = \zeta(1/2) + \zeta'(0) * d\tau$.
- Then we compute $\pi(1)$, from $\pi(0)$ and $\zeta'(1/2)$, $\Phi'(1/2)$ and $\Phi(1/2)$,

So in general, we have,

- First we compute $\zeta_{n+1/2} = \zeta_{n-1/2} + \zeta'(n)$, where in $\zeta'(n)$ we have everything at step n which we know the values!
$$\zeta_{n+\frac{1}{2}} = \zeta_{n-\frac{1}{2}} + \zeta'_n \Delta\tau \tag{26}$$
ζ'_n in the equation 26 reads from the differential equation as following,
$$\zeta'_n = 3\mathcal{H}_n(w\zeta_n + c_s^2\Psi_n) - c_s^2(3\mathcal{H}_n^2 - 3\mathcal{H}'_n)\pi_n + 3c_s^2\Phi'_n + c_s^2\nabla^2\pi_n \tag{27}$$
$$+ 2c_s^2\Phi_n\nabla^2\pi_n - (1-c_s^2)\Psi_n\nabla^2\pi_n - 3c_s^2\mathcal{H}_n(1+w)\pi_n\nabla^2\pi_n + (1-c_s^2)(\zeta_n+\Psi_n)\nabla^2\pi_n$$
$$- c_s^2\nabla\Phi_n.\nabla\pi_n + (2c_s^2-1)\nabla\Psi_n.\nabla\pi_n - \frac{\mathcal{H}_n}{2}\Big(2+3w+c_s^2\Big)\nabla\pi_n.\nabla\pi_n + 2(1-c_s^2)\nabla\pi_n.\nabla(\zeta_n+\Psi_n) \tag{28}$$
Where we only need $\Phi(n)$, $\Phi'(n)$, $\pi(n)$, $\zeta(n) = \frac{\zeta(n-1/2)+\zeta(n+1/2)}{2}$. We have all the quantities simply at step n, so we do not need any extrapolation and etc.
So by $\zeta(n) = \frac{\zeta(n-1/2)+\zeta(n+1/2)}{2}$ we again can factorize and for the $\nabla(\frac{\zeta_{(n-1/2)}+\zeta_{(n+1/2)}}{2})$ which can be factorized we again use predictor-corrector method by guessing in the first loop that it is simply $\nabla\zeta_{(n-1/2)/2}$
- Then we compute $\zeta(n+1)$ which is synchronized with everything by $\zeta(n+1) = \zeta(n) + \zeta'(n)d\tau$, we have defined two fields one is ζ_{int} and the other ζ_{half}. But note that it is better to update it by $\zeta_{n+1} = \zeta_n + \zeta'_{n+1/2}d\tau$, but in this case we need to have $\Phi'(n+1/2) = \frac{\Phi(n+1)-\Phi(n)}{d\tau}$ and of course we need $\Phi(n+1)$ which we don't have access to.
Lets for the moment calculate $\zeta_{n+1} = \zeta_n + \zeta'_{n+1/2}d\tau$ but take $\Phi'(n+1/2) = \Phi'(n)$ approximately which we guess is not a very far idea, but to be completely consistent its better to save three Φ and extrapolate to get $\Phi(n+1)$... but lets do the easiest for the moment! and if the solution is changed with increasing precision it means that our approximation is not very good!
- After updating $\zeta(n+1/2)$, we update π from n to n+1 as following,
$$\pi_{n+1} = \pi_n + \pi'_{n+\frac{1}{2}}\Delta\tau \tag{29}$$

$$\pi'_{n+\frac{1}{2}} = \zeta_{n+\frac{1}{2}} - \mathcal{H}_{n+\frac{1}{2}}\pi_{n+\frac{1}{2}} + \Psi_{n+\frac{1}{2}} \tag{30}$$

According to the definition, since the equation for updating π is linear we do not have any trouble with non-linear terms and this part is done like linear equations.
The background part is better to be updated before this step to have a_{kess} at (n+1/2) and then having $\mathcal{H}(n+1/2)$. Just note that making ζ updating at half steps help us because we need all the terms in $\zeta'(n)$ at integer steps.
Since the scalar field Stress energy tensor must be synchronized with particles stress tensor, we need to have all the variables at the same step which is n, so we need to write all the terms at step $n+\frac{1}{2}$ in terms of the values at step n and $n+1$ as following, of course except ζ which we have it at $n+1/2$. The easiest model to calculate $F_{n+\frac{1}{2}}$ is by taking average of the next and last step, so

$$\pi_{n+\frac{1}{2}} = \frac{\pi_{n+1} + \pi_n}{2} \tag{31}$$

and the same for all the other variables at step $n+\frac{1}{2}$.
For π we have,

$$\pi_{n+1} = \frac{1}{1+\mathcal{H}_{n+\frac{1}{2}}\Delta\tau/2}\left[\pi_n + \Delta\tau\left[\zeta_{n+\frac{1}{2}} - \mathcal{H}_{n+\frac{1}{2}}\frac{\pi_n}{2} + \Psi_{n+\frac{1}{2}}\right]\right] \tag{32}$$

As it is clear from the formula we don't have access to the $\Psi_{n+\frac{1}{2}}$ and Ψ_{n+1} , so we use the extrapolation to have them in next half steps,

$$\Psi_{n+\frac{1}{2}} = \Psi_n + \Psi'_n\frac{d\tau}{2} \tag{33}$$

Moreover to have Ψ'_n we use the following formula by saving Ψ at two different steps!

$$\Psi'_n = \frac{\Psi_n - \Psi_{n-1}}{d\tau} \tag{34}$$

As it is clear we do not have any trouble with π updating too, while in previous version of updating we were updating wrongly and ζ was ahead of potentials and we were using the wrong value for potentials.

To summerize:

- First loop: $\Phi(0)$, ζ_{half} (-1/2), $\zeta_{int}(0)$
- At the end of the first loop we update particles and kessence fields $\zeta_{half}(1/2)$, $\zeta_{int}(1)$, $\pi(1)$ and particles at step 1 .
  .
  .
- $T_{\mu\nu}(n)$ from particles and kessence at step n $\longrightarrow$ $\Phi(n), \chi(n)$ from stress tensor $\longrightarrow$ $\zeta_{half}(n+1/2), \zeta_{int}(n+1) \longrightarrow \pi(n+1) \longrightarrow$ particles move to step n+1

Another note that should be taken into account is that:
Since we have $\pi(n+1) = \pi(n) + \pi'(n+1/2)$, and in $\pi'(n+1/2)$ we have H(n+1/2) we update the background before updating the pi to have H(n+1/2), Moreover $\zeta(n+1) = \zeta(n+1/2)+\zeta'(n+1/2)$, so we put ζ_{int} updating in the pi updating, which is after background updating! So in the zeta updating we ζ_{half} updating and in the π updating we have both π

and ζ_{int} updating.
We also have saved three potentials in each step $\Phi(n)$, $\Phi(n-1)$ and $\Phi(n-2)$, so simply we have,

$$\Phi'(n) = \frac{\Phi(n) - \Phi(n-1)}{d\tau} \quad \Phi'(n-1) = \frac{\Phi(n-1) - \Phi(n-2)}{d\tau} \quad \Phi''(n) = \frac{\Phi'(n) - \Phi'(n-1)}{d\tau} \tag{35}$$

So we have,

$$\Phi'(n+1/2) = \Phi'(n-1/2) + \Phi''(n)d\tau. \tag{36}$$

so,

$$\Phi'(n+1/2) = \Phi'(n) + \frac{\Phi'(n) - \Phi'(n-1)}{d\tau}d\tau/2 = \frac{3}{2}\Phi'(n) - \Phi'(n-1)/2 \tag{37}$$

So simply by saving three potentials we can consistently compute everything we need, specially $\Phi'(n+1/2)$
A great point: In this case of updating although we are updating two ζ one $\zeta half$ and ζ_{int} but for $\zeta half$ updating in the $\zeta'(n)$ in the non-linear part we have $\nabla(\Psi(n) + \zeta(n))$ which we have $\zeta(n)$ from previous step so we do not need predictor-corrector method! Another discussion with Julian is,

> No. I was asking where you update \Phi(n) -> \Phi(n+1) in the loop you discussed. This is certainly done after the particles.
> So I do not understand the situation. In the main loop I'm using the last action is particles updating.

Ah, I now see your misconception. The loop count of gevolution does actually not strictly correspond to n. Let me explain.

The leapfrog of gevolution (without K-essence) is actually:

particle momenta(n-1/2) -> particle momenta(n+1/2)
particle positions(n) -> particle positions(n+1)
compute stress-energy at (n+1) by particle-mesh projection
metric(n) -> metric(n+1)

However, the main loop begins with "compute stress-energy". This means that gevolution is first completing the integer time step from the *previous* leapfrog step before it starts the next one. Of course it does not matter at all where the loop count is incremented in the above scheme, but I see that it may be confusing that the loop count is not equal to n.

Let me therefore write the above leapfrog equivalently so that n *is* the loop count:

compute stress-energy at n by particle-mesh projection
metric(n-1) -> metric(n)
particle momenta(n-1/2) -> particle momenta(n+1/2)
particle positions(n) -> particle positions(n+1)

At the end of the main loop, the particles are at (n+1) but the metric is just at n, waiting to be updated first thing at the next loop iteration.

Hope this clears things up.

No Still I have a problem!
to update $\zeta_{int}(n+1)$ from $\zeta(n)$ and $\zeta'(n+1/2)$ we need $\pi(n+1/2)$, $\nabla\pi(n+1/2)$ and etc, which is a big mess! Because

1.4 Non-linear terms tests: How?

Here we aim to check the non-linear terms alone, just to make sure everything is correctly computed or we did not forgot any trivial point. What we are going to do is solving the below two equations in mathematica and Gevolution to see if we get reasonable results! We believe that these two equations are enough to be solved, since all other terms look like one

of these two terms and also we have already checked that ζ and π up to a great order agree in mathematica and Gevolution, but maybe it would be good check to put all the possible terms together to see if we can solve in mathematica?

$$\pi' = \zeta + \Psi - \mathcal{H}\pi \tag{38}$$

$$\zeta' = \mathcal{H}\nabla\pi.\nabla\pi \tag{39}$$

$$\zeta' = c_s^2\Phi(\nabla\pi)^2 \tag{40}$$

Unfortunately, I do not know how to solve in matehamtica to compare with the result of Gevolution!

1.5 Predictor-corrector method

Here we try to measure the effect of non-linear terms. Before doing it we first check if predictor-corrector method works well?
To do, we measure how many times it goes through predictor-corrector method to converge? After doing some experiments, it seems that it works well! So for demanding precision like 10^{-10} which is actually the relative improvement, it just go through the loop once mostly or twice and some times specially low redshifts it must go 10 times in the loop to reach the precision, so it hits precision error in low redshifts.
As an example when we ask for the the relative precision of 10^{-8} and ask if the numerator was more than 6 print the precision we get the following result in the terminal,

```
numerator: 7 The rel diff: 7.00482e-10
numerator: 7 The rel diff: 8.28997e-10
numerator: 8 The rel diff: 6.95646e-10
numerator: 7 The rel diff: 6.65728e-10
numerator: 7 The rel diff: 3.09033e-09
numerator: 7 The rel diff: 1.9231e-09
numerator: 7 The rel diff: 3.23853e-09
numerator: 8 The rel diff: 6.69198e-10
numerator: 7 The rel diff: 1.74146e-09
numerator: 7 The rel diff: 6.87903e-10
numerator: 7 The rel diff: 7.36749e-10
numerator: 7 The rel diff: 2.34831e-09
numerator: 8 The rel diff: 6.397e-10
numerator: 7 The rel diff: 3.06782e-09
numerator: 7 The rel diff: 1.82822e-09
numerator: 7 The rel diff: 7.01182e-10
```

The numerator is the number of times which go through corrector loop to improve the variable! For the more realistic case we consider getting near to the previous one to 10^{-4} relative difference which is less than 1%, we see that it goes at most 4 times over the loop and if we ask for printing the values of numerator more than 4 we get,

```
cycle 90, background information: z = 1.85694, average T00 = 0.312045, background model = 0.312046
 cycle 90, time integration information: max |v| = 0.00339536 (cdm Courant factor = 0.0648959), time step
     / Hubble time = 0.04
 numerator: 5 The rel diff: 1.0313e-05
 numerator: 5 The rel diff: 2.32914e-05
 writing power spectra at z = 0.996995 (cycle 99), tau/boxsize = 18.0884
 cycle 100, background information: z = 0.918918, average T00 = 0.312044, background model = 0.312046
 cycle 100, time integration information: max |v| = 0.00415543 (cdm Courant factor = 0.087711), time step
     / Hubble time = 0.04
 numerator: 5 The rel diff: 1.71626e-05
 cycle 110, background information: z = 0.285983, average T00 = 0.312043, background model = 0.312046
 cycle 110, time integration information: max |v| = 0.00598645 (cdm Courant factor = 0.124585), time step
     / Hubble time = 0.04
 numerator: 5 The rel diff: 6.9317e-06
 numerator: 5 The rel diff: 1.67579e-05
 numerator: 5 The rel diff: 1.45638e-05
```

Which is just four times and at low redshifts! We also do not see numerator more than 5!

1.6 Non-linear contribution in the code

To measure the non-linear terms contribution, we have couple of ways. One is comparing the the linear versus non-linear contribution plot on different variables...
The other one is just internally look at the relative importance of non-linear terms in comparison with linear terms, so in ζ' we just measure the relative between non-linear terms and linear ones $\frac{\text{all non-linear contributions}-\text{all linear contributions}}{\text{all linear contributions}}$, which is as following,

$$\frac{\Big(+2c_s^2\Phi_{n+1}\nabla^2\pi_{n+1}-(1-c_s^2)\Psi_{n+1}\nabla^2\pi_{n+1}\Big)...\Big(-3\mathcal{H}_{n+1}(w\frac{\zeta_{n+\frac{1}{2}}}{2}+c_s^2\Psi_{n+1})-c_s^2(3\mathcal{H}_{n+1}^2-3\mathcal{H}'_{n+1})\pi_{n+1}+...\Big)}{3\mathcal{H}_{n+1}(w\frac{\zeta_{n+\frac{1}{2}}}{2}+c_s^2\Psi_{n+1})-c_s^2(3\mathcal{H}_{n+1}^2-3\mathcal{H}'_{n+1})\pi_{n+1}+3c_s^2\Phi'_{n+1}+c_s^2\nabla^2\pi_{n+1}} \tag{41}$$

what we get is somehow strange! So in different points of lattice we get the following terminal output for the quantity which shows the importance of non-linear terms!

```
NL_REL: -2.36611
NL_REL: -2.38774
NL_REL: -2.29923
NL_REL: -2.22916
NL_REL: -2.18889
NL_REL: -2.15739
NL_REL: -2.31122
NL_REL: -2.48264
NL_REL: -2.54007
NL_REL: -2.4611
NL_REL: -2.47478
NL_REL: -2.38359
NL_REL: -2.35794
NL_REL: -2.38377
NL_REL: -2.4141
NL_REL: -2.33127
NL_REL: -2.29721
NL_REL: -2.33786
NL_REL: -2.33795
NL_REL: -2.35654
NL_REL: -2.30935
NL_REL: -2.24938
NL_REL: -2.30449
NL_REL: -2.4566
NL_REL: -2.37959
NL_REL: -2.64718
NL_REL: -2.40469
NL_REL: -2.30812
NL_REL: -2.44066
NL_REL: -2.42957
NL_REL: -2.24973
NL_REL: -2.48907
NL_REL: -2.41429
NL_REL: -2.40282
```

the top results are related to the relative importance of non-linear terms in each point! The physical quantity we are interested in is the powerspectrum which is measured below.