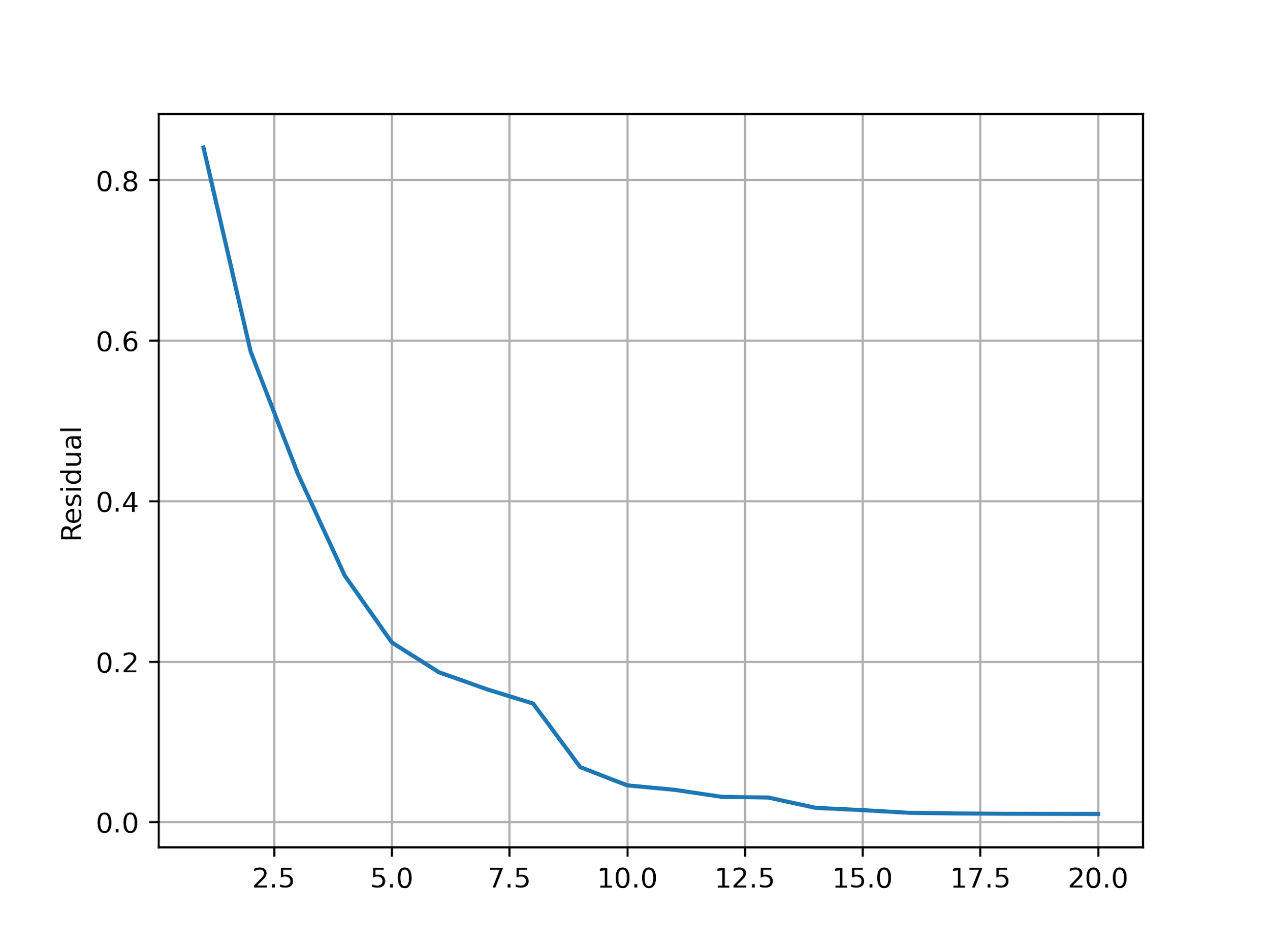
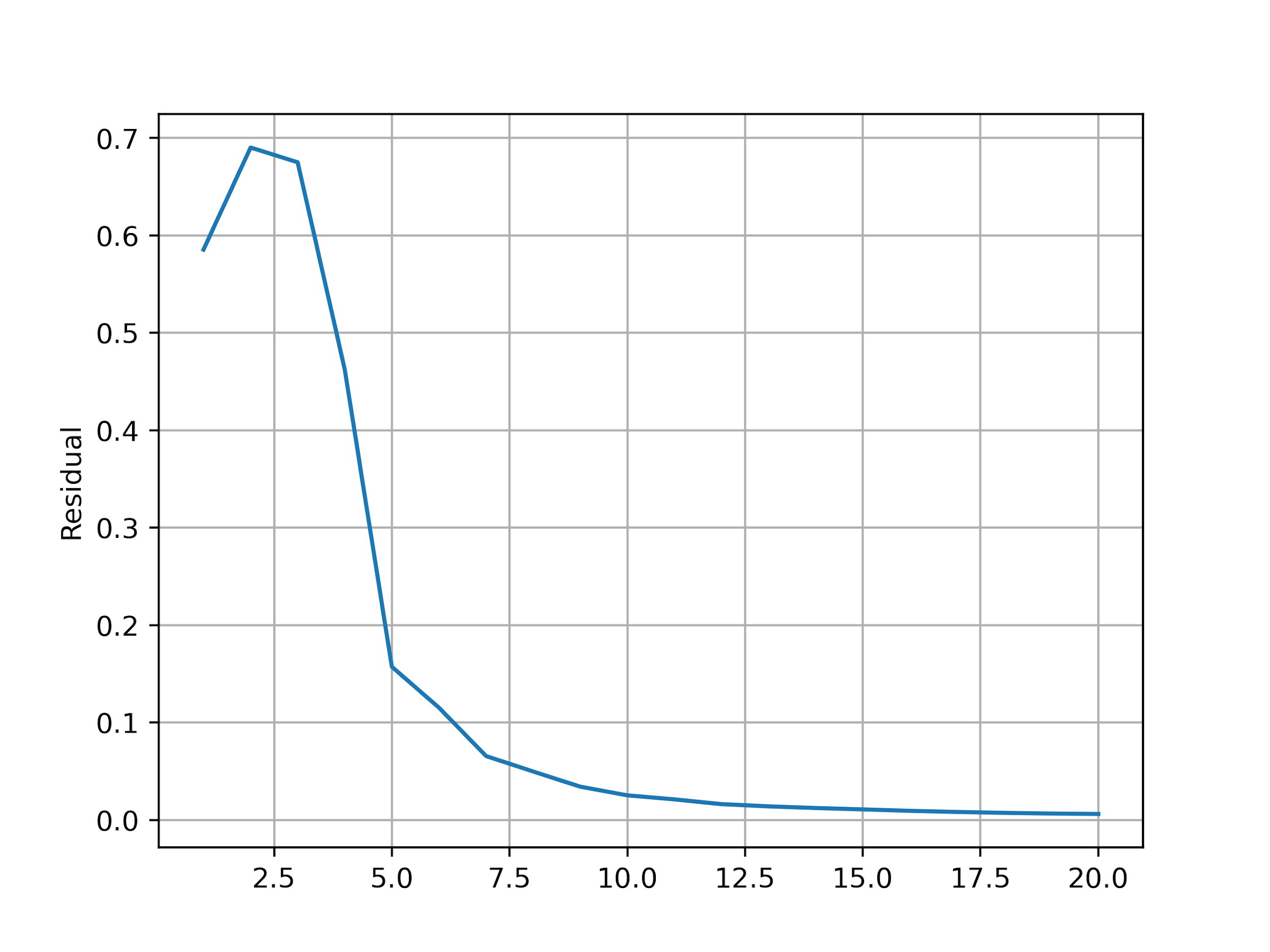
**About convergence**

The User Story #15276 states that the algorithms are not converging to zero, but with the correct parameters the algorithms can converge to zero. To achieve convergence to zero FiniteDifferenceForwardModel has to be used and when using ConjugateGradient the parameter n\_max has to be set to 1 (such that forwardModel::calculateKappa is called only once) while iter1.n is used to control the maximum amount of iterations (for ConjugateGradient the maximum amount of iterations is n\_max \* iter1.n).

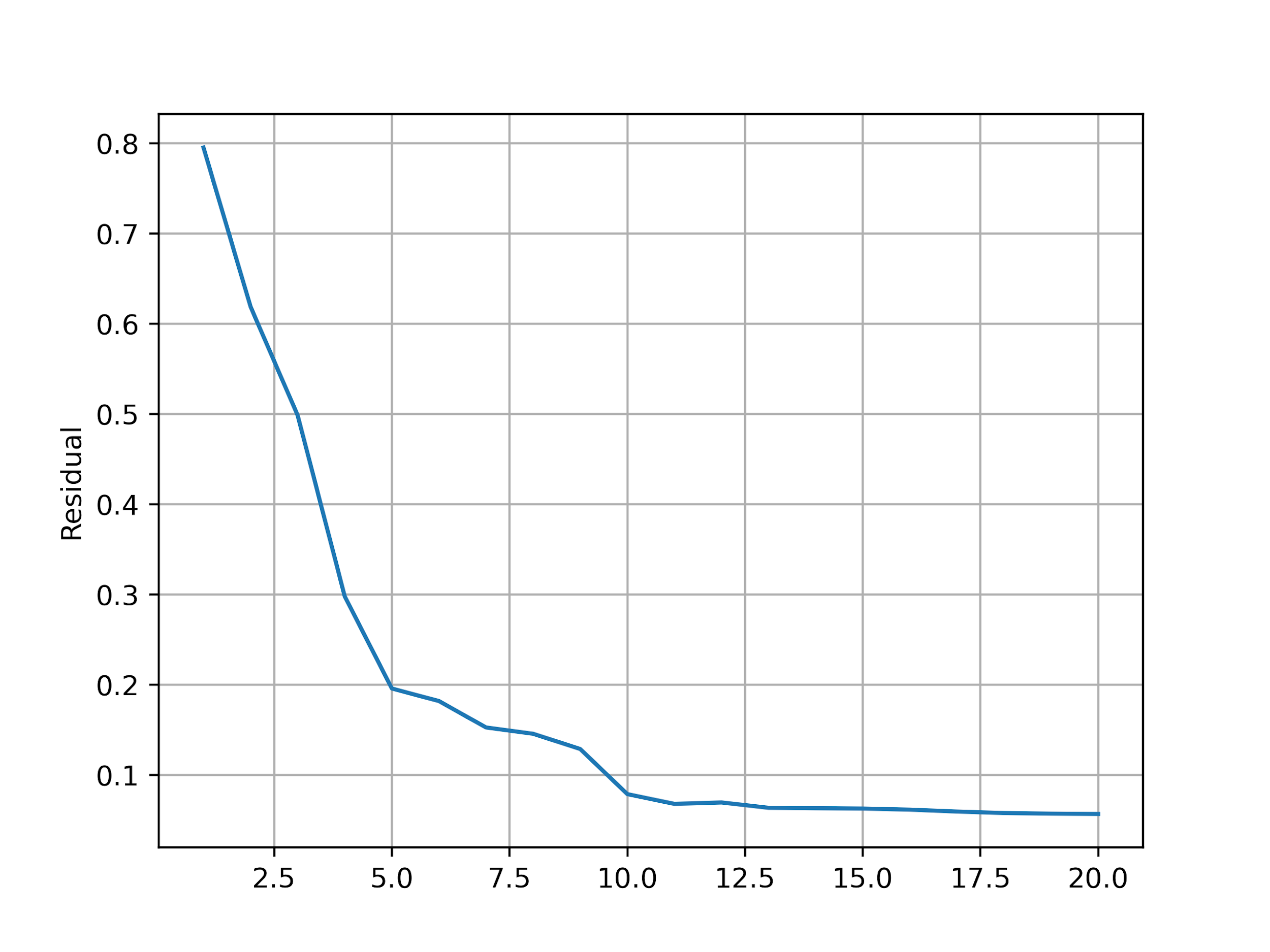


**Figure 1**: GradientDescent “Residual” using FiniteDifferenceForwardModel.

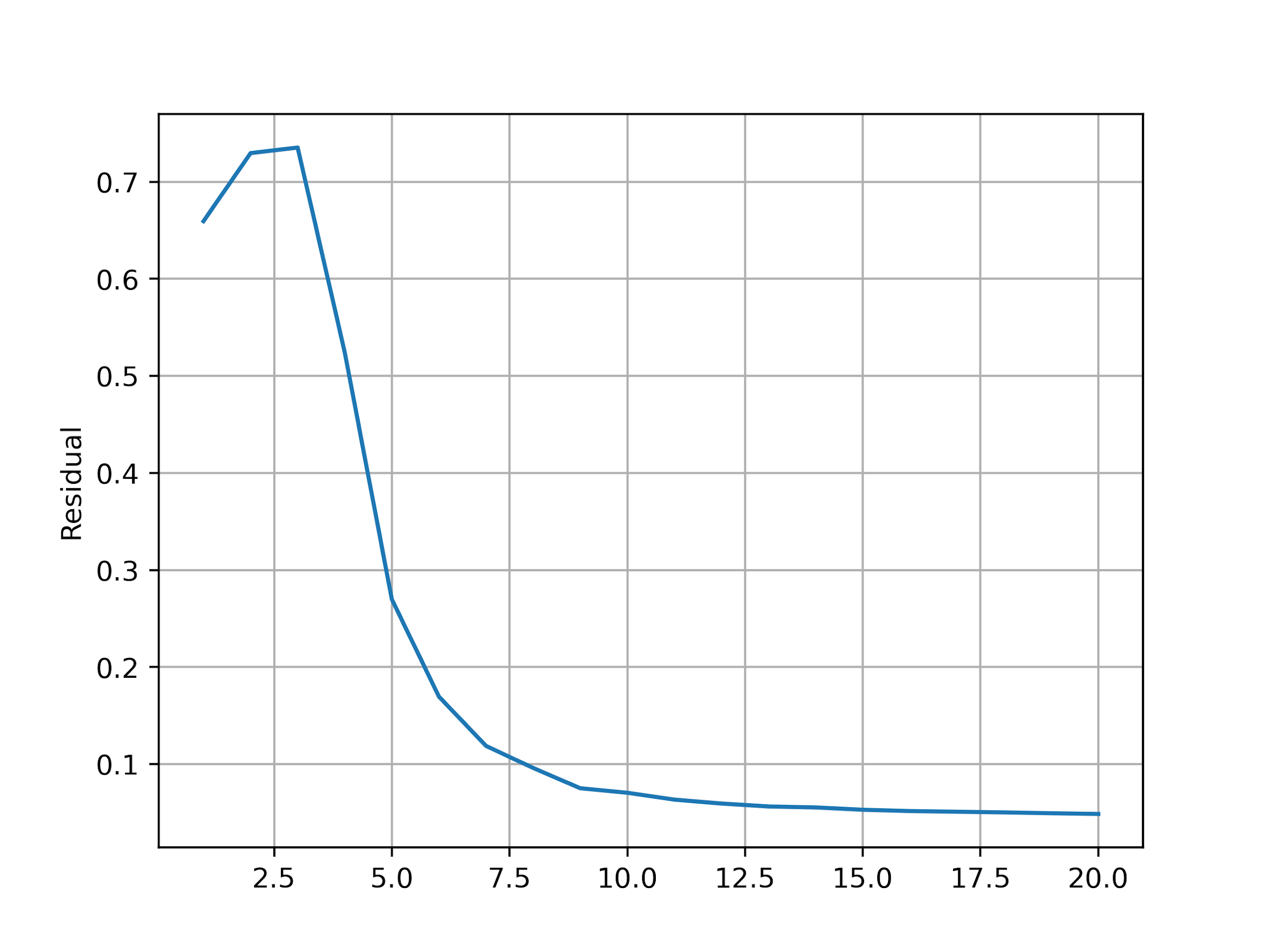


**Figure 2**: ConjugateGradient “Residual” using FiniteDifferenceForwardModel.

When using IntegralForwardModel convergence to zero can't be achieved.



**Figure 3**: GradientDescent “Residual” using IntegralForwardModel.



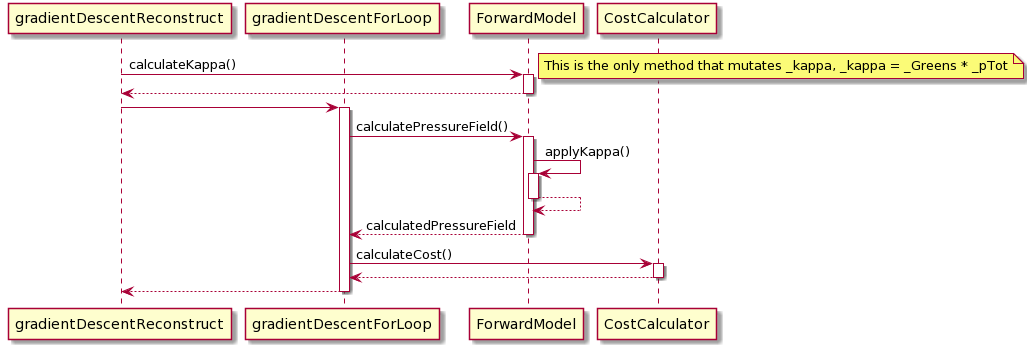
**Figure 4**: ConjugateGradient “Residual” using IntegralForwardModel.

Conclusions:

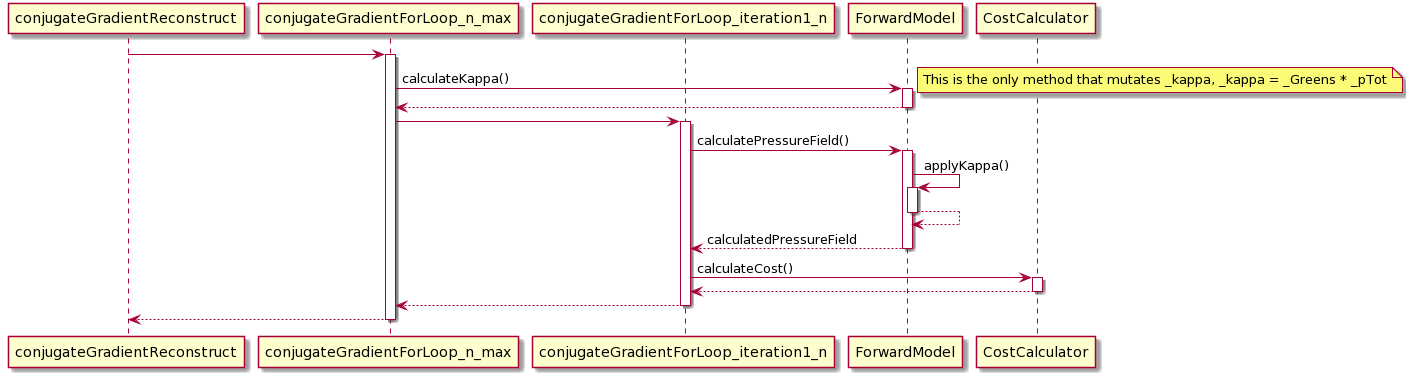
* IntegralForwardModel needs to be investigated on why it is not converging to zero.
* GradientDescent can take hours to finish running, we need to investigate why.
* Selecting n\_max > 1 causes ConjugateGradient to call calculateKappa n\_max times, this gives worse results than setting n\_max == 1 for all possible input images so it seems to hint that this is an implementation mistake but further investigation is needed, please check the attached test results.

**How is the residual computed and what are the “Residual” graphs**

The residual is a vector which in all cases is equal to the referencePressureField - pressureFieldObtainedFromCurrentMediaPropertiesEstimation, and we use this value to calculate the cost which is the scalar value that we end up optimizing. When looking at the “Residual” graphs we need to keep in mind that “Residual” actually should be named Cost since what we are looking at is the cost calculated by the CostCalculator using the residual. Here are the sequence diagrams that show how the cost is obtained.



**Figure 5**: Sequence diagram for GradientDescent



**Figure 6**: Sequence diagram for ConjugateGradient

In order to refer to how the inversion methods use the forwardModels we need to take a look at the functions in the forwardModelInterface:

* getUpdateDirectionInformationMPI not used by anyone, should be removed.
* getUpdateDirectionInformation is used only by ConjugateGradient::calculateUpdateDirection, which is needed to obtain the new estimate of the media properties, we should think if it is possible to move the functionality of getUpdateDirectionInformation to ConjugateGradient in order to decouple the forwardModel from this particular optimization method.
* calculateKappa, the results from the tests suggest that this function should only be called only once so this function should be removed from the interface and its functionality should be moved to each of the ForwardModels constructors.
* calculatePtot is only used by the preProcessing applications and by randomInversion, more investigation is needed to determine why this is needed.

**What are the important values on the forwardModels to calculate the pressure field from an estimate of the media properties**

The ForwardModels have 4 members that determine the calculated pressure field, they are \_kappa,\_pTot,\_p0,\_Greens. These are allocated and initialized in the constructor of each forwardModel. Some comments about them:

* \_Greens , is the impulse response of the system and it remains constant after ForwardModel construction.
* \_p0, initial guess on the pressure field remains constant after ForwardModel construction.
* \_pTot, total pressure field as it would be recorded by receivers, it remains equal to \_p0 after ForwardModel construction unless calculatePTot is called afterwards.
* \_kappa, it is needed to calculate a new pressure field from a current estimate of the media properties ( newPressureField = \_kappa \* currentMediaPropertiesGuess ), and it remains constant after ForwardModel construction unless calculateKappa is called afterwards.