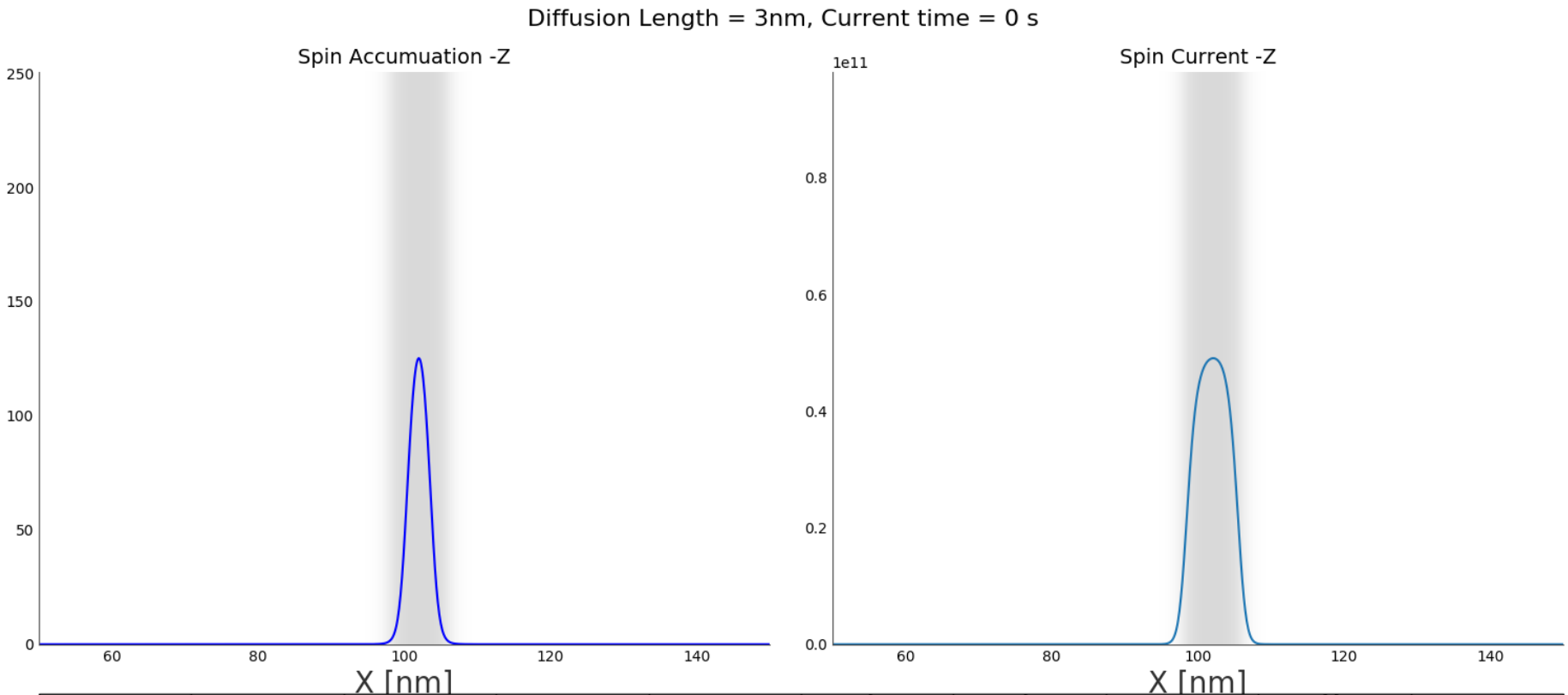
Some results of the new method...

Small time step -1×10^{-21}s

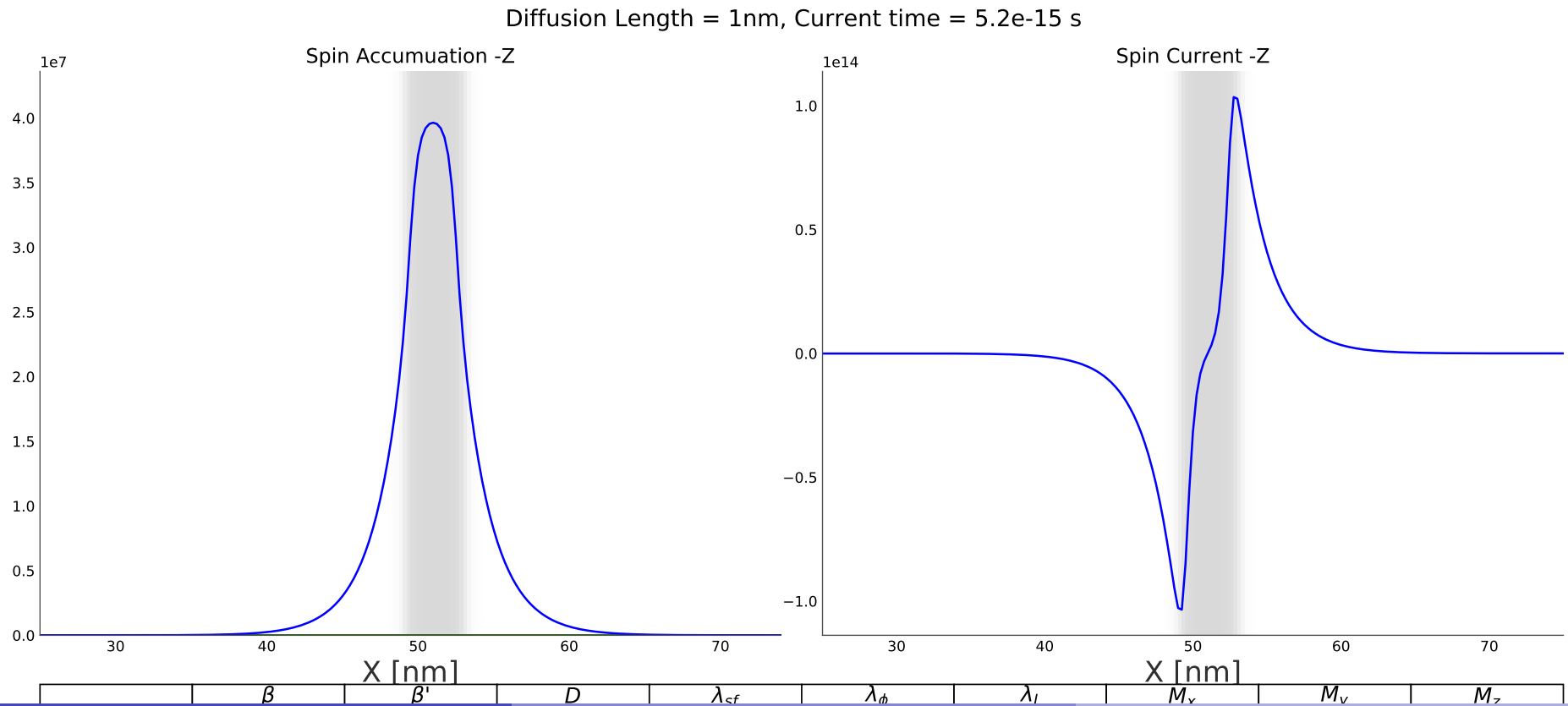
We can see that the spin current evolves from being initially just a hump due to the $\beta(x)\hat{M}j_e$ term to resembling the derivative of a gaussian when the $\frac{\partial\vec{m}(x,t)}{\partial x}$ term starts to dominate. (Click to see the animation)



	β	β'	D	λ_{sf}	λ_{ϕ}	λ_J	M_x	M_y	M_z
Non-Magnet	0	0	0.005	7e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0	0	0.005	7e-08	1e-20	4e-20	0	0	0

Reaching Equilibrium

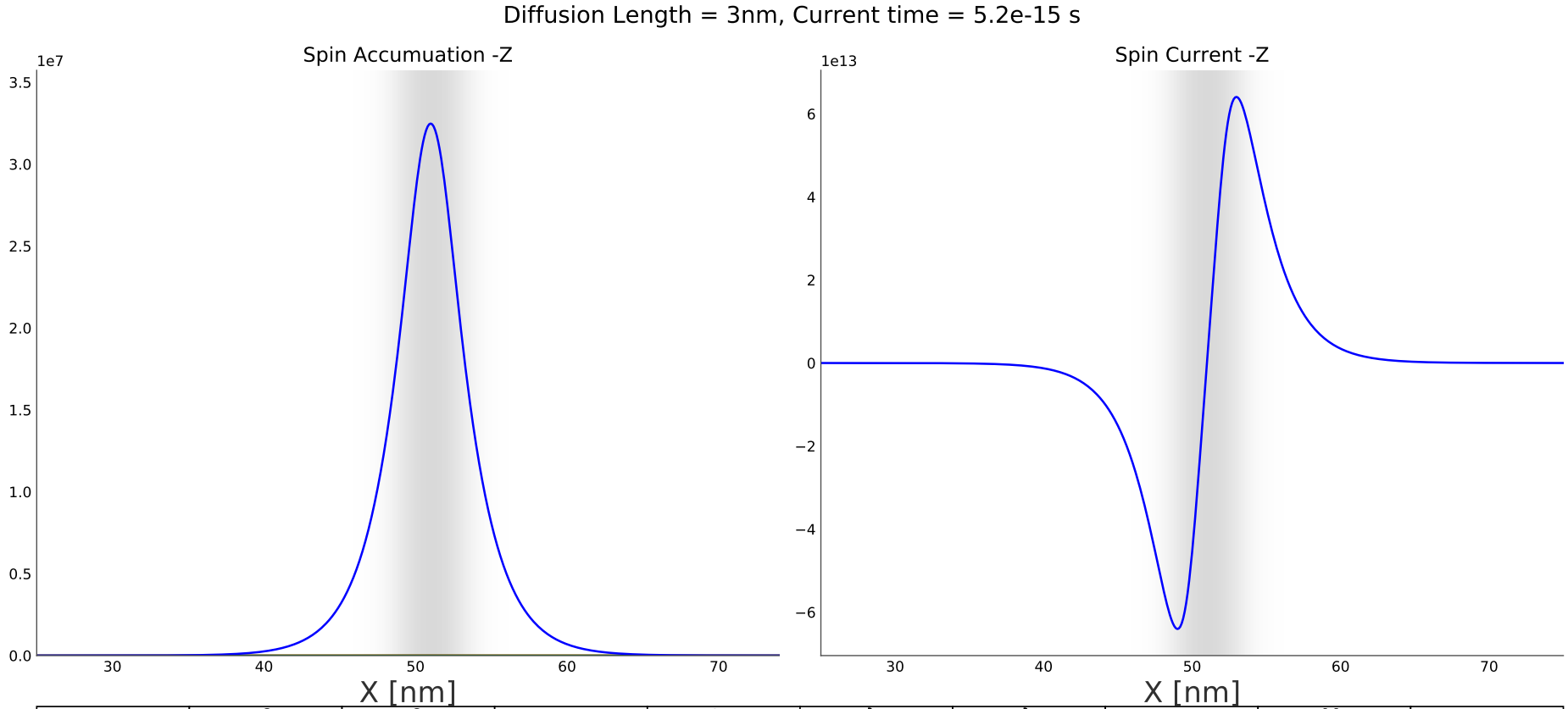
I left this simulation to reach equilibrium using a time step of 1×10^{-17}s. It has seemed to reach the m_∞ value of 4×10^7 but I'm not sure if it has spread out normally into the non-magnet. λ_{sf} is 20nm in the non-magnet and it has only spread 20 nm, I think it should spread more? The picture is a link to an animation showing the simulation relaxing into equilibrium.



Sharper Interfaces

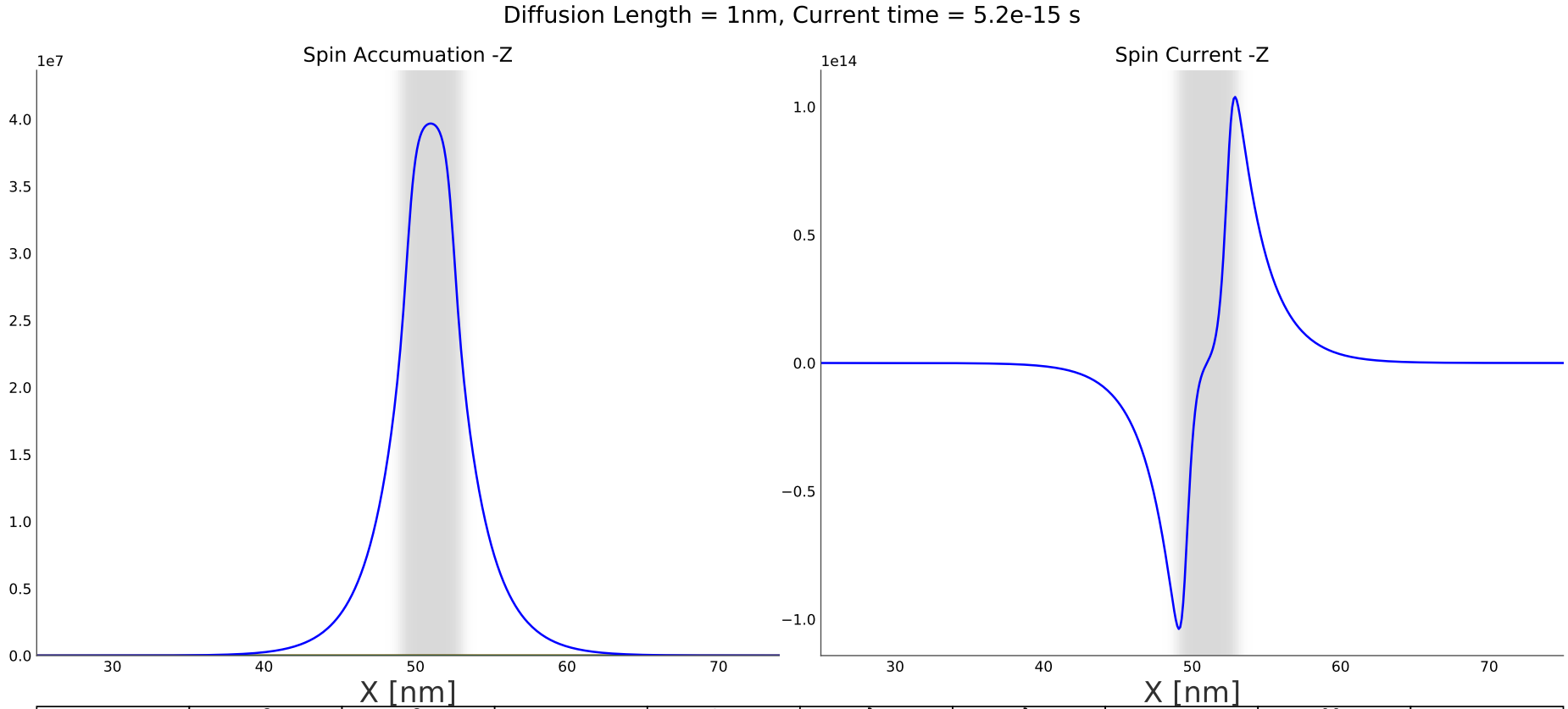
In the previous simulations I used a diffusion length of $\sim$1 nm. This controls the diffusiveness of the interfaces. The next 3 slides show simulations ran to equilibrium for different DLs. We can see the spikes in the spin current getting sharper for decreasing DL implying that for perfectly sharp interfaces the gradient of the spin current would have a discontinuity. I'm not sure about the actual spin current though.

DL = 3nm



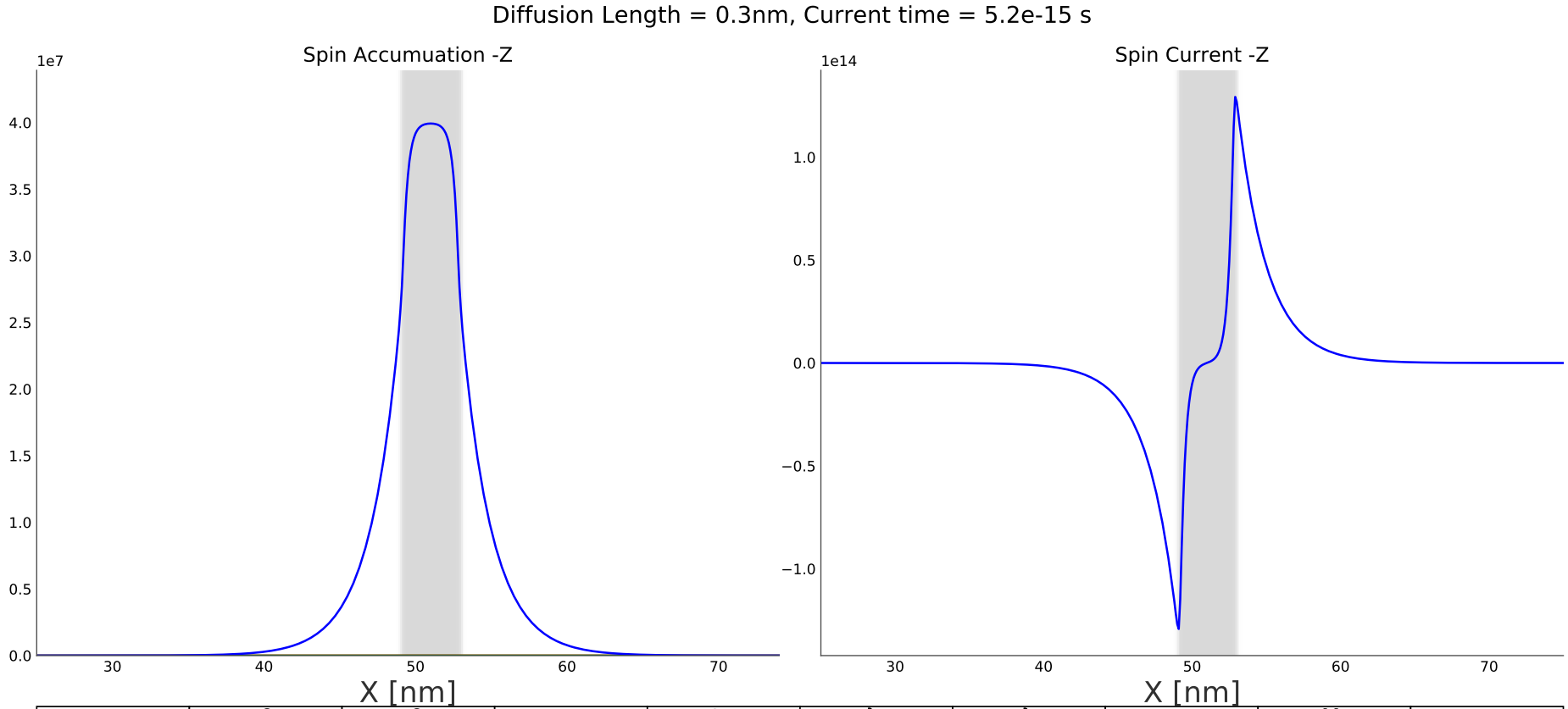
	β	β'	D	λ_{sf}	λ_{ϕ}	λ_J	M_x	M_y	M_z
Non-Magnet	0	0	0.005	2e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0	0	0.005	2e-08	1e-20	4e-20	0	0	0

DL = 1nm



	β	β'	D	λ_{sf}	λ_{ϕ}	λ_J	M_x	M_y	M_z
Non-Magnet	0	0	0.005	2e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0	0	0.005	2e-08	1e-20	4e-20	0	0	0

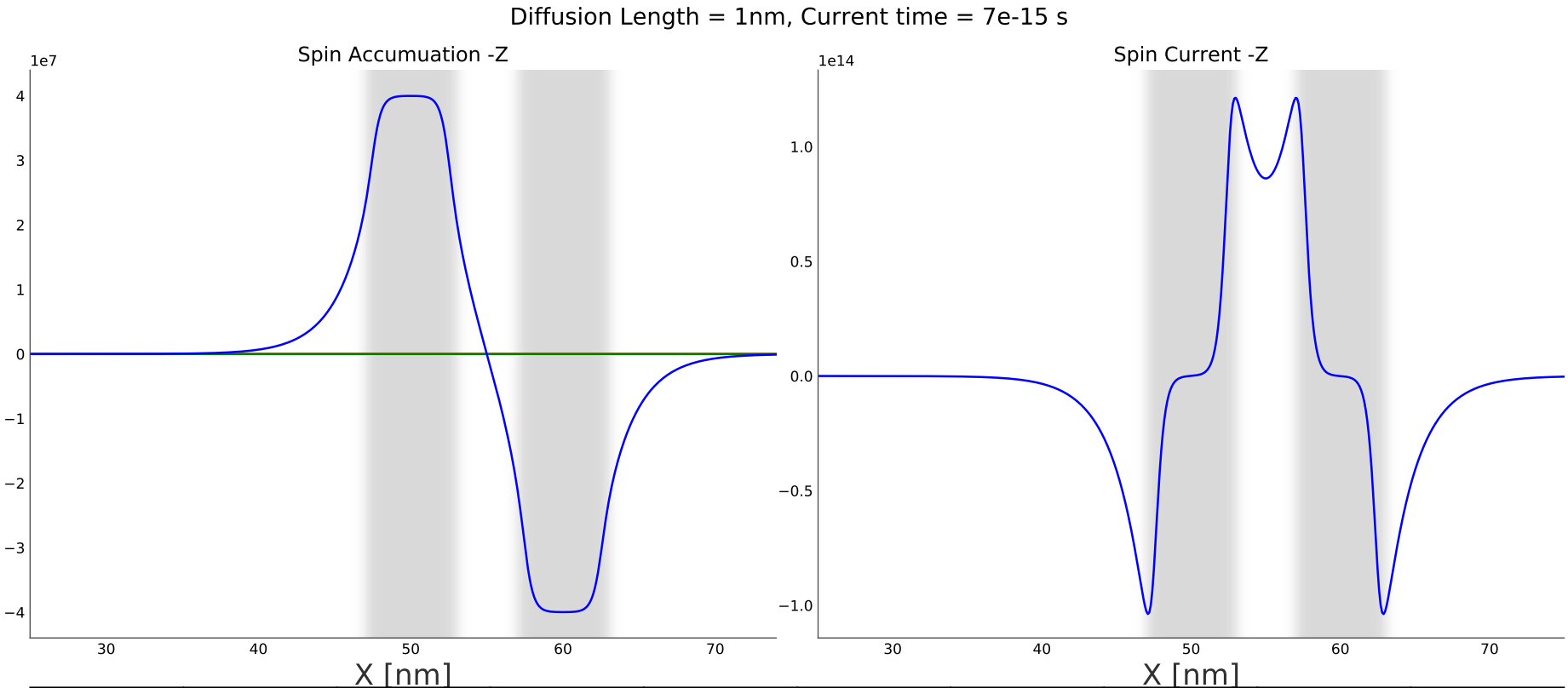
DL = 0.3nm



	β	β'	D	λ_{sf}	λ_{ϕ}	λ_J	M_x	M_y	M_z
Non-Magnet	0	0	0.005	2e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0	0	0.005	2e-08	1e-20	4e-20	0	0	0

2 Layers

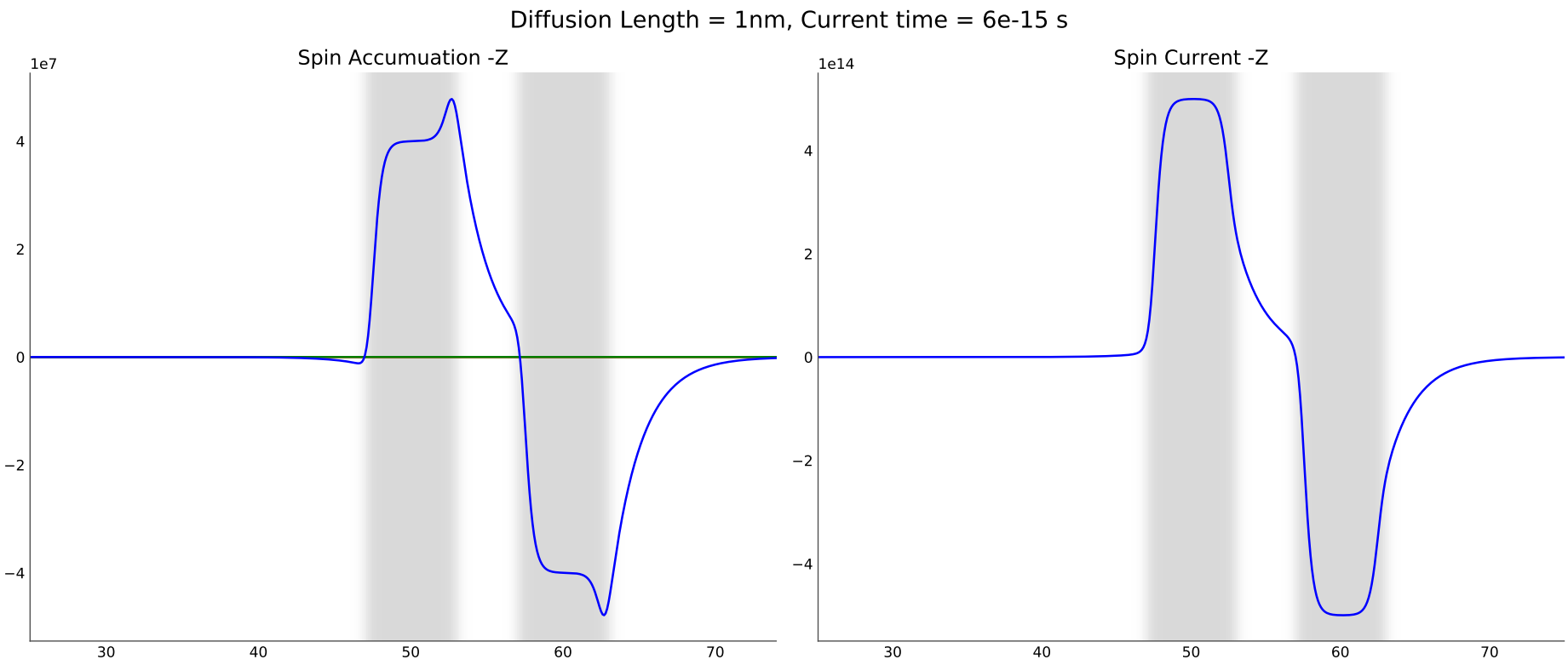
Here is a graph showing the results of 2 layers (only Z).



	β	β'	D	λ_{sf}	λ_{ϕ}	λ_{J}	M_x	M_y	M_z
Non-Magnet	0	0	0.005	2e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	-1

Large Charge Current

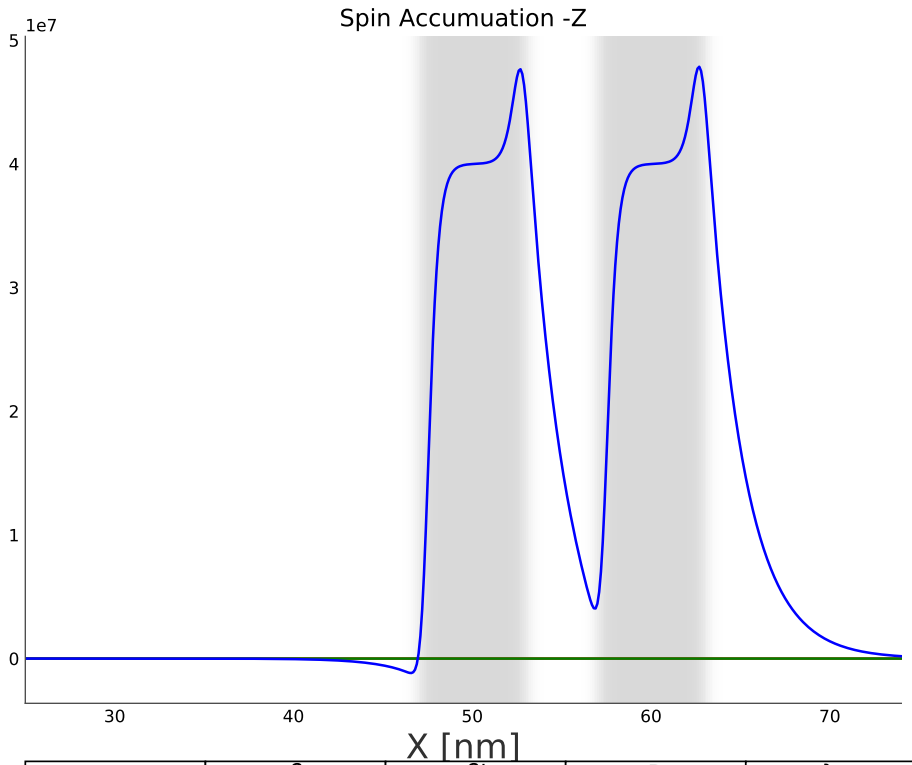
Richard mentioned a while ago that the charge current may lead to a the spin current being 'pushed' to the right. The simulation seems to show this for large charge currents $\sim 10^{15}$ click the following image to see it evolve.

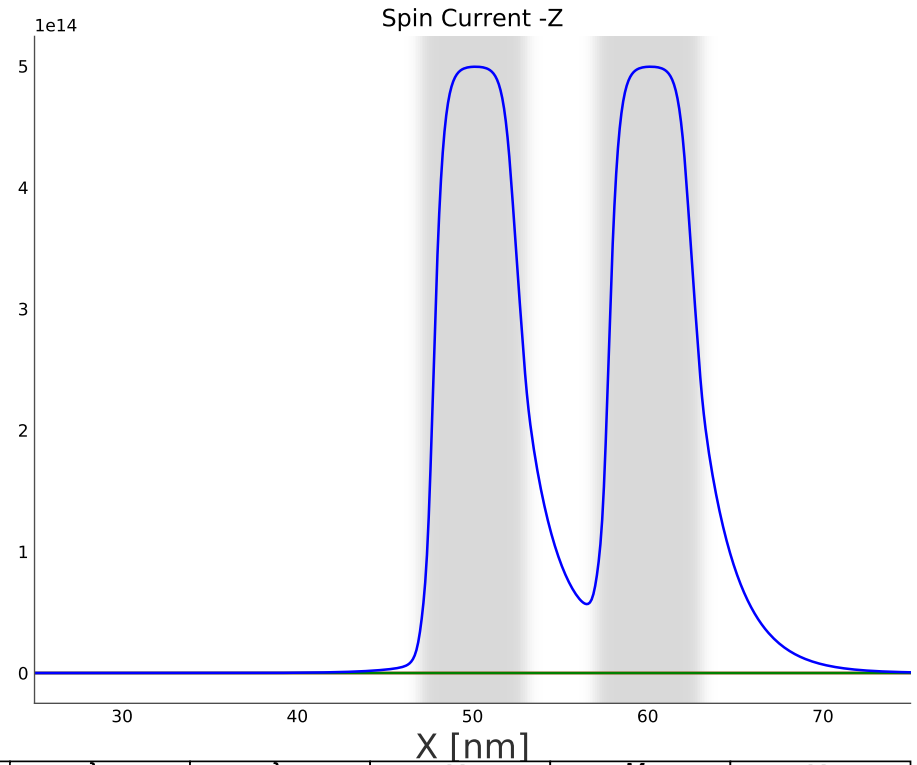


	β	β'	D	λ_{sf}	λ_{ϕ}	λ_J	M_x	M_y	M_z
Non-Magnet	0	0	0.005	2e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	-1

Parallel Configuration

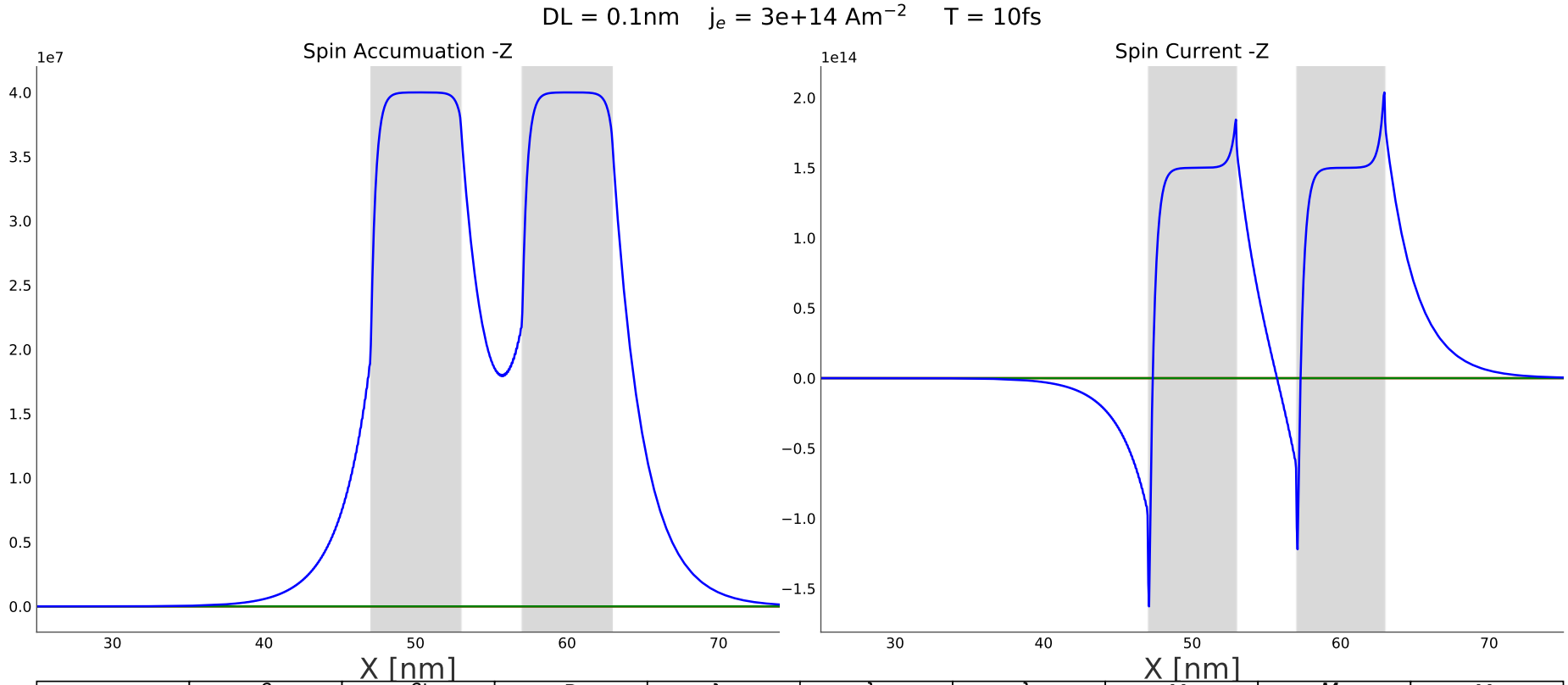
DL = 1nm j_e = 1e+15 Am^{-2} T = 10fs





	β	β'	D	λ_{sf}	λ_{ϕ}	λ_J	M_x	M_y	M_z
Non-Magnet	0	0	0.005	2e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1

Discontinuities appear with large charge current and perfectly sharp interfaces.

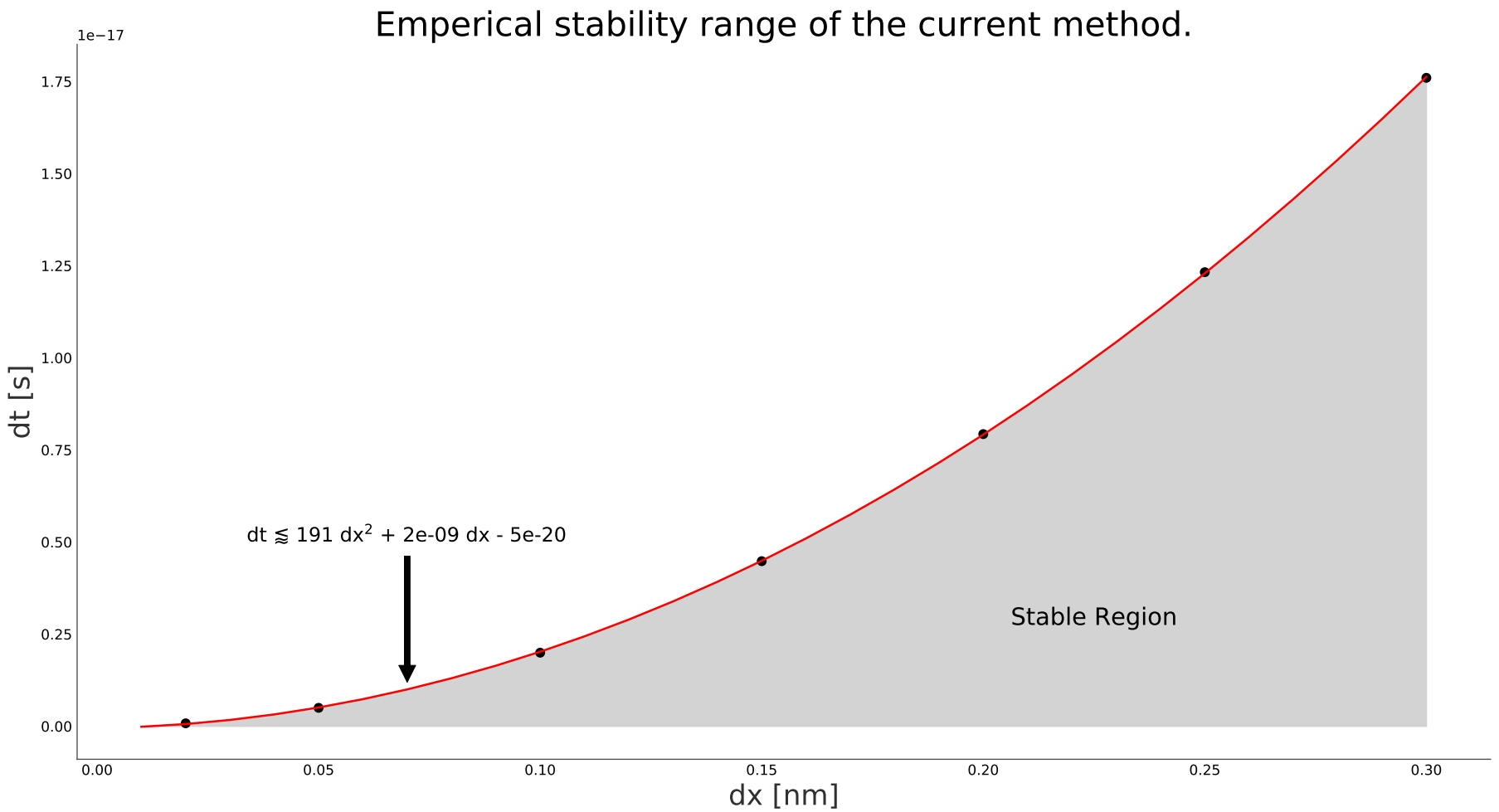


	β	β'	D	λ_{sf}	λ_{ϕ}	λ_J	M_x	M_y	M_z
Non-Magnet	0	0	0.005	2e-08	1e-20	1e-20	0	0	0
F1	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1
F2	0.5	0.7	0.003	5e-09	4e-09	4e-09	0	0	1

If I use other components of magnetisation the simulation seems to become unstable. The same thing happens in the C++ code I wrote from scratch which probably means its unlikely to be a bug in the code?

Stability Criteria (without using λ_J or λ_ϕ)

After some running the code and stopping it when it seems unstable I've come up with the empirical stability criteria:



I think the code becomes unstable when I use other magnetisations as this introduces new terms to the equation (the λ_J & λ_ϕ terms)...

Using a value of 10 nm for λ_J and dt $= 10^{-28}$s the code can't get to a time of T $= 10^{-24}$ without being unstable.

However, if I lower the time step to dt $= 10^{-29}$s the code is stable until T $= 10^{-24}$.

When I heighten the λ terms the code becomes more stable.

Maybe this suggests I need a new more stable method.

Still to do

- Find out why the code is unstable when other magnetisations are used? Even in the C++ code this is the case. Maybe this is actually a problem with the method rather than a bug in the code?
- Probably sensible to implement some testing software! My code is very bug prone!
- Calculate the resistance for parallel and anti-parallel configurations (can't do other at the moment)! See if that checks out i.e. anti-parallel resistance $<$ parallel resistance?