NU Provides an Energy-Injection Mechanism

A. Chawla

August 1, 2025

Abstract

In this note the authors investigate vorticity in liquid helium under the traditional Landau-Tisza model as well as a modified model. It is shown that the information approach allows injection of energy into the vortex which leads to a rising kinetic energy and oscillating momentum.

Key Insight: The note includes detailed MATLAB code (Algorithms 1-4) that simulates vortex dynamics in liquid helium with adaptive parameters, nonlocal terms, and informational density evolution. The work blends established fluid dynamics with speculative theoretical physics, proposing that information fields could serve as a bridge between consciousness and physical reality in cosmological contexts.

In fundamental physics, the universe begins from vacuum wherein there arises a spontaneous fluctuation (with some probability). Modern theory is not equipped to probe beyond this instance (cf. the recent interview of Prof. Henry Tye [URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC8363324/]). However, nonlocal unification (NU hereafter) provides a mechanism whereby an information field can affect spacetime. In this note we show that in the case of liquid helium.

The script simulates a vortex in liquid helium under the standard Landau theory. At the same time, we modify the script to allow the injection of energy

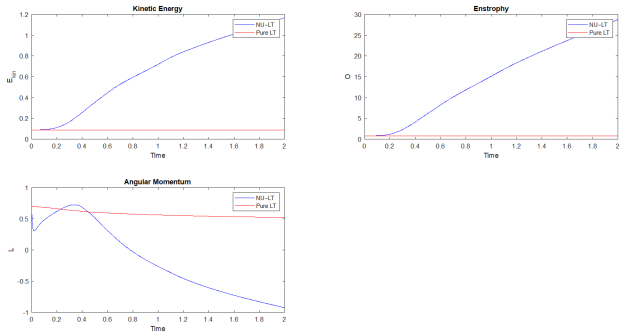

Figure 1: Vortex dynamics modified via informational density evolution.

Algorithm 1 Energy Injection via Informational Density Evolution - I

```
% landauTisza_nu_navstokes.m
% Extended simulation of Liquid Helium dynamics with NU-informed Navier-Stokes framework
clear; clc; close all;
% === Simulation Parameters ===
Nx = 256; Ny = 256; % Grid size (power of 2 for FFT)
dx = 0.1; dy = 0.1; % Spatial step
dt_base = 0.002; % Base time step
steps = 1500; % Number of steps
rho_total = 1.0; % Total density
nu_0 = 0.01; % Base viscosity
kappa = 1.0; % Circulation quantum
alpha = 0.1; % Nonlocal kernel strength
sigma = 0.5; % Nonlocal kernel width
gamma = 0.01; % Informational density diffusion
alpha_nu = 0.1; % Viscosity modulation strength (Appendix P)
% === Spatial Grid ===
x = ((-Nx/2):(Nx/2-1)) * dx;
y = ((-Ny/2):(Ny/2-1)) * dy;
[X, Y] = meshgrid(x, y);
k = 2*pi*fftshift((-Nx/2:Nx/2-1)/(Nx*dx)); % Fourier wavenumbers
[Kx, Ky] = meshgrid(k, k);
K2 = Kx.^2 + Ky.^2; K2(1,1) = 1; % Avoid division by zero
% === Nonlocal Kernel ===
r2 = X.^2 + Y.^2;
G = alpha * exp(-r2/(2*sigma^2)) / (2*pi*sigma^2); % Gaussian kernel
G = G / sum(G(:)); % Normalize
G_hat = fft2(G); % FFT of kernel
% === Initial Conditions ===
% Informational density I(x, y)
I = 1.0 + 0.03 * exp(-r2/2);
I_max = max(I(:));
% Densities
rho_s_nu = rho_total * 0.9 .* (I / I_max);
rho_s_pure = rho_total * 0.9 * ones(size(I));
rho_n = rho_total - rho_s_nu;
% Initial vortex ring
vx_init = -Y .* exp(-r2 / 0.5);
vy_init = X .* exp(-r2 / 0.5);
% Initialize velocity fields
vx_s_nu = vx_init; vy_s_nu = vy_init;
vx_n_nu = zeros(size(X)); vy_n_nu = zeros(size(X));
vx_s_pure = vx_init; vy_s_pure = vy_init;
vx_n_pure = zeros(size(X)); vy_n_pure = zeros(size(X));
% === Analysis Storage ===
E_kin_nu = zeros(steps, 1); E_kin_pure = zeros(steps, 1); % Kinetic energy
Omega_nu = zeros(steps, 1); Omega_pure = zeros(steps, 1); % Enstrophy
L_nu = zeros(steps, 1); L_pure = zeros(steps, 1); % Angular momentum
% === Plotting Setup ===
figure('Color', 'w', 'Position', [100, 100, 1200, 500]); % Increased height for 3 subplots
subplot(2,2,1);
h1 = imagesc(x, y, zeros(Nx, Ny)); colorbar;
axis equal tight; title('NU–LT Vorticity'); xlabel('x'); ylabel('y');
```

Algorithm 2 Energy Injection via Informational Density Evolution - II

```
subplot(2,2,2);
h2 = imagesc(x, y, zeros(Nx, Ny)); colorbar;
axis equal tight; title('Pure LT Vorticity'); xlabel('x'); ylabel('y');
subplot(2,2,3);
h3 = imagesc(x, y, I); colorbar;
axis equal tight; title('Informational Density I'); xlabel('x'); ylabel('y');
subplot(2,2,4);
h4 = plot(0, 0, 'b'); hold on; % Initialize angular momentum plot
axis tight; title('Angular Momentum'); xlabel('Time'); ylabel('L');
legend('NU–LT', 'Pure LT');
drawnow;
% === Helper Functions ===
% FFT-based Laplacian
laplacian = @(Z) real(ifft2(-K2 .* fft2(Z)));
% FFT-based Poisson solver for projection
poisson_solve = @(f) real(ifft2(-fft2(f) ./ K2));
% Project velocity to divergence-free field
project = @(vx, vy) deal(...
vx - real(ifft2(1i*Kx .* fft2(poisson_solve(real(ifft2(1i*Kx.*fft2(vx) +
1i*Ky.*fft2(vy)))))), ...
vy - real(ifft2(1i*Ky .* fft2(poisson_solve(real(ifft2(1i*Kx.*fft2(vx) +
1i*Ky.*fft2(vy)))))));
% Compute vorticity
vorticity = @(vx, vy) real(ifft2(1i*Ky.*fft2(vx) - 1i*Kx.*fft2(vy)));
% Nonlocal convolution
nonlocal_term = @(f) real(ifft2(G_hat .* fft2(f)));
% Adaptive viscosity based on vorticity gradient (Appendix P)
compute_nu_eff = @(omega) nu_0 * (1 + alpha_nu * abs(gradient(omega,
dx, dy)));
% Adaptive time step based on informational density (Appendix P)
f_delta_I = @(delta_I) 1 ./ (delta_I.^3 + 0.1); % Nonlinear time modulation
compute_dt = @(I) dt_base * f_delta_I(abs(gradient(I, dx, dy)));
% Absorbing boundary layer
damp = ones(Nx, Ny);
edge = 20;
for i = 1:Nx
for j = 1:Ny
if abs(x(i)) > max(abs(x))-edge*dx || abs(y(j)) > max(abs(y))-edge*dy
damp(i,j) = exp(-((max(abs(x(i))-max(abs(x))+edge*dx,0)/(edge*dx))^2));
end
end
end
% === Main Loop ===
for step = 1:steps
% Compute adaptive time step
dt = min(compute_dt(I), [], 'all'); % Use minimum dt for stability
% === NU–LT Update ===
% Update informational density I
I = I + dt * (gamma * laplacian(I) + 0.5 * (vx_s_nu.^2 + vy_s_nu.^2));
I = max(I, 0.1); % Prevent negative I
rho_s_nu = rho_total * 0.9 .* (I / max(I(:)));
rho_n = rho_total - rho_s_nu;
```

Algorithm 3 Energy Injection via Informational Density Evolution - III

```
% Compute vorticity for adaptive viscosity
omega_nu = vorticity(vx_s_nu, vy_s_nu);
nu_eff = compute_nu_eff(omega_nu);
% Nonlocal terms
nl_x = nonlocal_term(vx_s_nu ./ (rho_s_nu + 1e-5));
nl_y = nonlocal_term(vy_s_nu ./ (rho_s_nu + 1e-5));
% Superfluid velocity update
[grad_Iy, grad_Ix] = gradient(I, dy, dx);
vx_s_nu = vx_s_nu + dt * (kappa * grad_Iy ./ (rho_s_nu + 1e-5) + nl_x);
vy_s_nu = vy_s_nu - dt * (kappa * grad_Ix ./ (rho_s_nu + 1e-5) - nl_y);
% Normal fluid velocity update with adaptive viscosity
vx_n_nu = vx_n_nu + dt * nu_eff .* laplacian(vx_n_nu);
vy_n_nu = vy_n_nu + dt * nu_eff .* laplacian(vy_n_nu);
% Project to divergence-free
[vx_s_nu, vy_s_nu] = project(vx_s_nu, vy_s_nu);
[vx_n_nu, vy_n_nu] = project(vx_n_nu, vy_n_nu);
% Apply damping
vx_s_nu = vx_s_nu .* damp; vy_s_nu = vy_s_nu .* damp;
vx_n_nu = vx_n_nu .* damp; vy_n_nu = vy_n_nu .* damp;
% === Pure LT Update ===
omega_pure = vorticity(vx_s_pure, vy_s_pure);
nu_eff_pure = compute_nu_eff(omega_pure); % Apply adaptive viscosity
here too
vx_n_pure = vx_n_pure + dt * nu_eff_pure .* laplacian(vx_n_pure);
vy_n_pure = vy_n_pure + dt * nu_eff_pure .* laplacian(vy_n_pure);
[vx_s_pure, vy_s_pure] = project(vx_s_pure, vy_s_pure);
vx_s_pure = vx_s_pure .* damp; vy_s_pure = vy_s_pure .* damp;
% === Analysis ===
E_kin_nu(step) = 0.5 * sum(sum(rho_s_nu .* (vx_s_nu.^2 + vy_s_nu.^2)
+ ...
rho_n .* (vx_n_nu.^2 + vy_n_nu.^2))) * dx * dy;
E_kin_pure(step) = 0.5 * sum(sum(rho_s_pure .* (vx_s_pure.^2 +
vy_s_pure.^2) + ...
(rho_total-rho_s_pure) .* (vx_n_pure.^2 + vy_n_pure.^2))) * dx * dy;
Omega_nu(step) = 0.5 * sum(sum(omega_nu.^2)) * dx * dy;
Omega_pure(step) = 0.5 * sum(sum(omega_pure.^2)) * dx * dy;
% Compute angular momentum for vortex ring (centered at origin)
mass = rho_s_nu * dx * dy; % Mass per cell (using superfluid density)
L_nu(step) = sum(sum(mass .* (X .* vy_s_nu - Y .* vx_s_nu))); % z-
component of angular momentum
L_pure(step) = sum(sum(mass .* (X .* vy_s_pure - Y .* vx_s_pure))); %
For pure LT
% === Update Plots ===
if mod(step, 50) == 0
set(h1, 'CData', omega_nu); caxis([-0.5, 0.5]);
set(h2, 'CData', omega_pure); caxis([-0.5, 0.5]);
set(h3, 'CData', I); caxis([min(I(:)), max(I(:))]);
set(h4, 'XData', dt_base*(1:step), 'YData', L_nu(1:step)); % Update NU–LT
angular momentum
hold on;
plot(dt_base*(1:step), L_pure(1:step), 'r'); % Update Pure LT angular mo-
mentum
```

Algorithm 4 Energy Injection via Informational Density Evolution - IV

```
hold off;
axis tight;
drawnow;
fprintf('Step %d / %d: E_kin_nu = %.4f, E_kin_pure = %.4f, dt = %.4f\n',
...
step, steps, E_kin_nu(step), E_kin_pure(step), dt);
end
end
% === Final Analysis Plot ===
figure('Color', 'w');
subplot(2,2,1);
plot(dt_base*(1:steps), E_kin_nu, 'b', dt_base*(1:steps), E_kin_pure, 'r');
legend('NU–LT', 'Pure LT'); title('Kinetic Energy'); xlabel('Time'); ylabel('E_{kin}');
subplot(2,2,2);
plot(dt_base*(1:steps), Omega_nu, 'b', dt_base*(1:steps), Omega_pure, 'r');
legend('NU–LT', 'Pure LT'); title('Enstrophy'); xlabel('Time'); ylabel('Ω');
subplot(2,2,3);
plot(dt_base*(1:steps), L_nu, 'b', dt_base*(1:steps), L_pure, 'r');
legend('NU–LT', 'Pure LT'); title('Angular Momentum'); xlabel('Time'); ylabel('L');
```

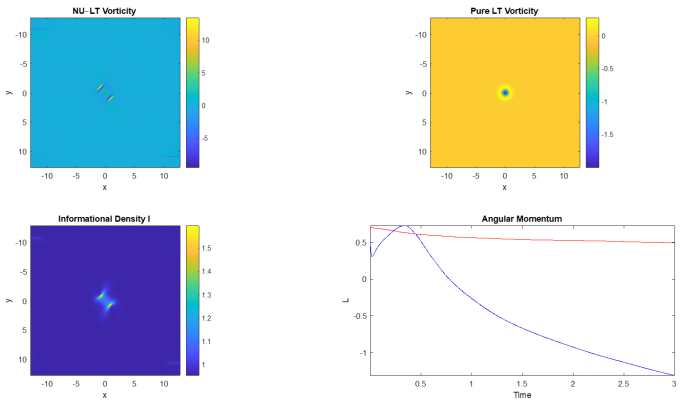

Figure 2: Visualization of vortex dynamics modified via informational density evolution.

via informational density evolution. This results in modulation of the vortex angular momentum and unbounded rise in its kinetic energy whereas the classical model shows energy conservation.

It was argued previously that consciousness does not interact with the physical universe. This normal scenario is possibly violated in the creation ex-nihilo setting. There information regulated injections of energy could have initiated the physical cosmos. The present work illuminates how this might have come about.