Ising Model of Ferromagnetism

The Ising model involves a lattice of N magnetic moments, S_i, which can take on discrete values ± 1, analogous to electron spins. The energy of element i in the lattice is

$$U_i = -BS_i - \sum_{j=1}^{N} J_{ij} S_i S_j,$$

where B is the externally applied magnetic field, and the coupling between moments is:

$$J_{ij} = \begin{cases} 1 & \text{for nearest neighbors,} \\ 0 & \text{otherwise.} \end{cases}$$

We define the magnetization, M, as

$$M = \langle S \rangle \equiv \frac{1}{N} \sum_{j=1}^{N} S_j.$$

By this definition, $-1 \leq M \leq 1$. As you can see, the model works in dimensionless units for both energy and magnetization.

My presentation here follows some old lecture notes from Paul Coddington at Syracuse University. I would provide a link but, alas, I can't find his materials on the web anymore.

Detailed Balance

The probability of a state change is derived from the concept of detailed balance as follows.

$$W(A \to B)\, P(A) = W(B \to A)\, P(B),$$

where A, B are among the possible states, W denotes the rate of the specified transition, and $P(A)$ is the fraction of things in state A. Assuming a Boltzmann distribution (with the partition function cancelling),

$$\frac{W(A \to B)}{W(B \to A)} = \frac{P(B)}{P(A)} = \frac{e^{-E(B)/kT}}{e^{-E(A)/kT}} = e^{-\Delta E/kT},$$

where $\Delta E = E(B) - E(A)$.

Metropolis Algorithm

The MonteCarlo model, as I will describe it below, involves an iterative application of random spin flips. For this, we need a criterion to determine whether the spin will flip. The temperature, kT, sets the probability that a spin S_j will change state at any given iteration. In agreement with the detailed balance calculation above, we choose

$$W(A \to B) = \begin{cases} e^{-\Delta E/kT} & \text{if} \Delta E > 0; \\ 1 & \text{if} \Delta E \leq 0. \end{cases}$$

This choice is arbitrary, but it produces the correct ratio of rates that we found in the previous section. Consequently, it is guaranteed to converge to the correct (Boltzmann) distribution as we iterate.

MonteCarlo Model

The Ising model with the Metropolis algorithm is generally implemented as a MonteCarlo model, that is, a program that simulates a physical system by repeated application of a random number generator. The standard approach is as follows:

1. Select a cell at random, and calculate ΔE.
2. Draw a random number R, from a parent distribution that is uniformly distributed between 0 and 1.
3. If $R > e^{-\Delta E/kT}$, then flip the spin.
4. Repeat.

Implementation

My implementation is a very simple Python module called `ferromagnet.py`. It includes two significant improvements to the model described above, one one physical (or at least physics-inspired), and one algorithmic.

Defects

It turns out that the Ising model of a ferromagnet, as described so far, likes to be magnetized. In fact, it likes it way too much. The sides of the hysteresis curve turn out to be quite vertical. This has two consequences that make the model behave unlike many of the ferromagnetic materials we encounter in daily life:

1. At finite temperature, the lattice gradually evolves toward complete magnetization. Many real ferromagnets tend to lose magnetization with time.
2. The process of AC demagnetization (which I demonstrate below) doesn't work. Let me explain: If you apply a strong, sinusoidally oscillating field to a real hunk of iron and gradually diminish the amplitude, it will demagnetize. The 2D Ising ferromagnet will instead wander toward high magnetization almost every time.

I have solved the above problems by intentionally putting defects in the lattice. These defects are spins at randomly chosen places within the lattice that are frozen either up or down, also at random. This is implemented via a defect map, which is a 2D Boolean array the same size as the spin lattice, with the defects marked **`True`** and all other locations marked **`False`**.

Smart Iteration Procedure

The standard approach requires something like for loop, selecting one spin at a time, randomly. This is slow for two reasons:

1. The iteration is typically done by a `for` loop, which is inefficient in an interpreted language, even a very efficient one such as Python.
2. Statistically, many spins will not be interrogated even after twice as many iterations as there are cells in the lattice. They just randomly get stuck for a little while.

The Smart iteration (named for grad student Roy Smart, who suggested it to me) is not just one iteration, but one for *every spin in the lattice*. We want to ensure that every spin is interrogated and allowed to flip exactly once, so that convergence can be obtained quickly. To do this, we go through the lattice in a special order. Imagining the lattice as a checkerboard, we do all the black squares first, treating them as a group (using Python array slicing and vectorization to speed things up), and then all the white squares. Notice that none of the black squares are nearest neighbors to each other; the same is true of the white squares.

The code, `ferromagnet.py`, is reasonably efficient because I have vectorized the process of performing this iteration. However, there is still considerable wasted effort in the execution. It could be speeded up by a factor of about 2 or more, but I don't care. As is my usual habit, I've struck a balance between simplicity (which goes along with readability) and efficiency. If efficiency were the only important thing, I would have coded it in C.

A Note on Running the Notebook

At a minimum, I recommend running once with 10 defects, and once with 1000 defects.

Even with a 101×101 lattice (my default size), the results of any one of the numerical experiments below may or may not be representative. The Ising model is a stochastic system, so the results will be different every time. If you want to estimate the behavior of a large, macroscopic system then it is necessary either to run with an extremely large lattice or run the experiment many, many times and average them.

In [1]:
```
import sys
import numpy as np
import matplotlib.pyplot as plt
# %matplotlib notebook
```

In [2]:
```
import ferromagnet as fe
```

In [3]:
```
# Randomly initialized Ising lattice
spins = fe.ising()
print('Magnetization = ', fe.magnetization(spins))
```

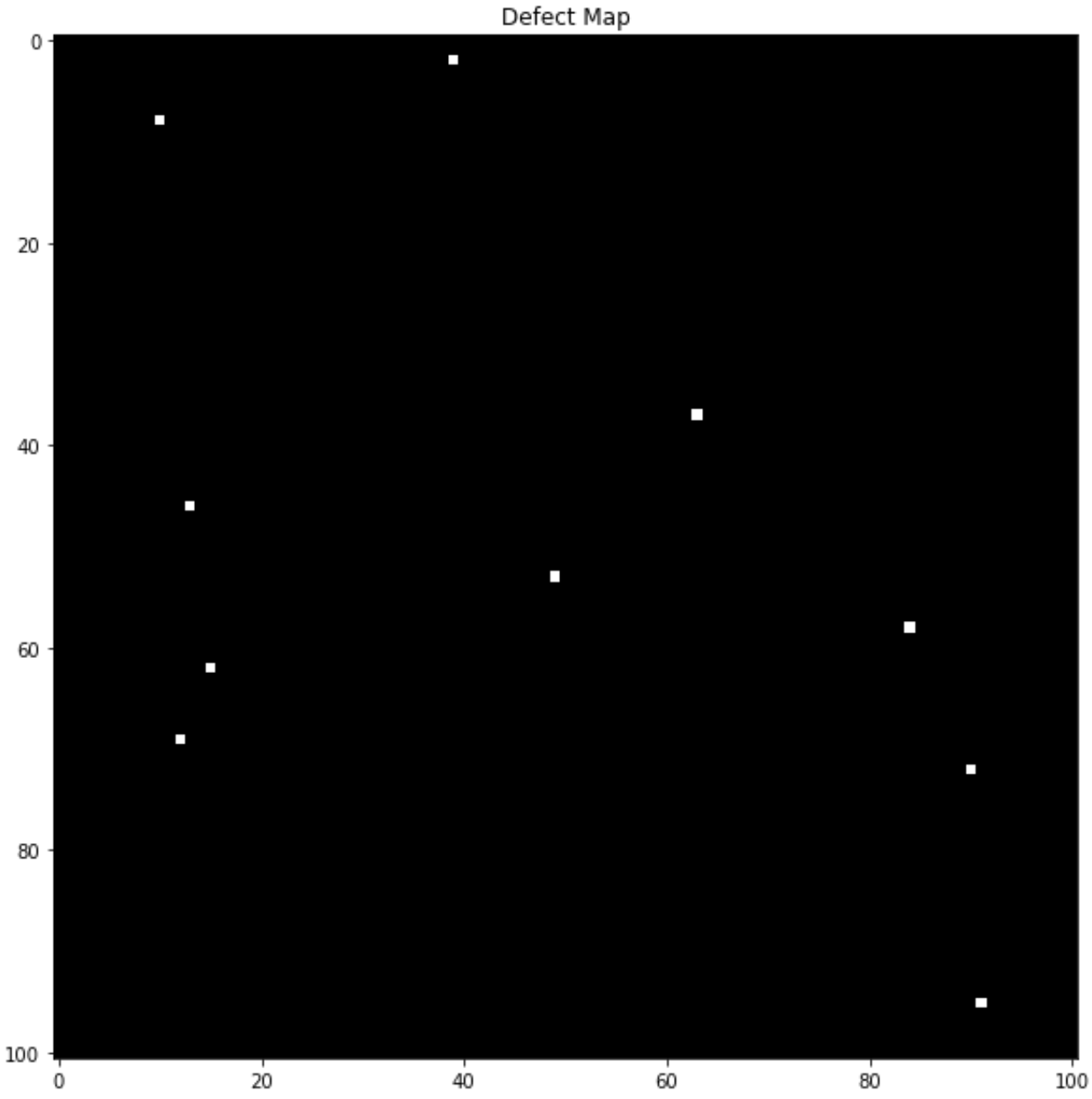
```
Ising model with  10201  cells.
```


```
Magnetization =  0.0173512400745025
```

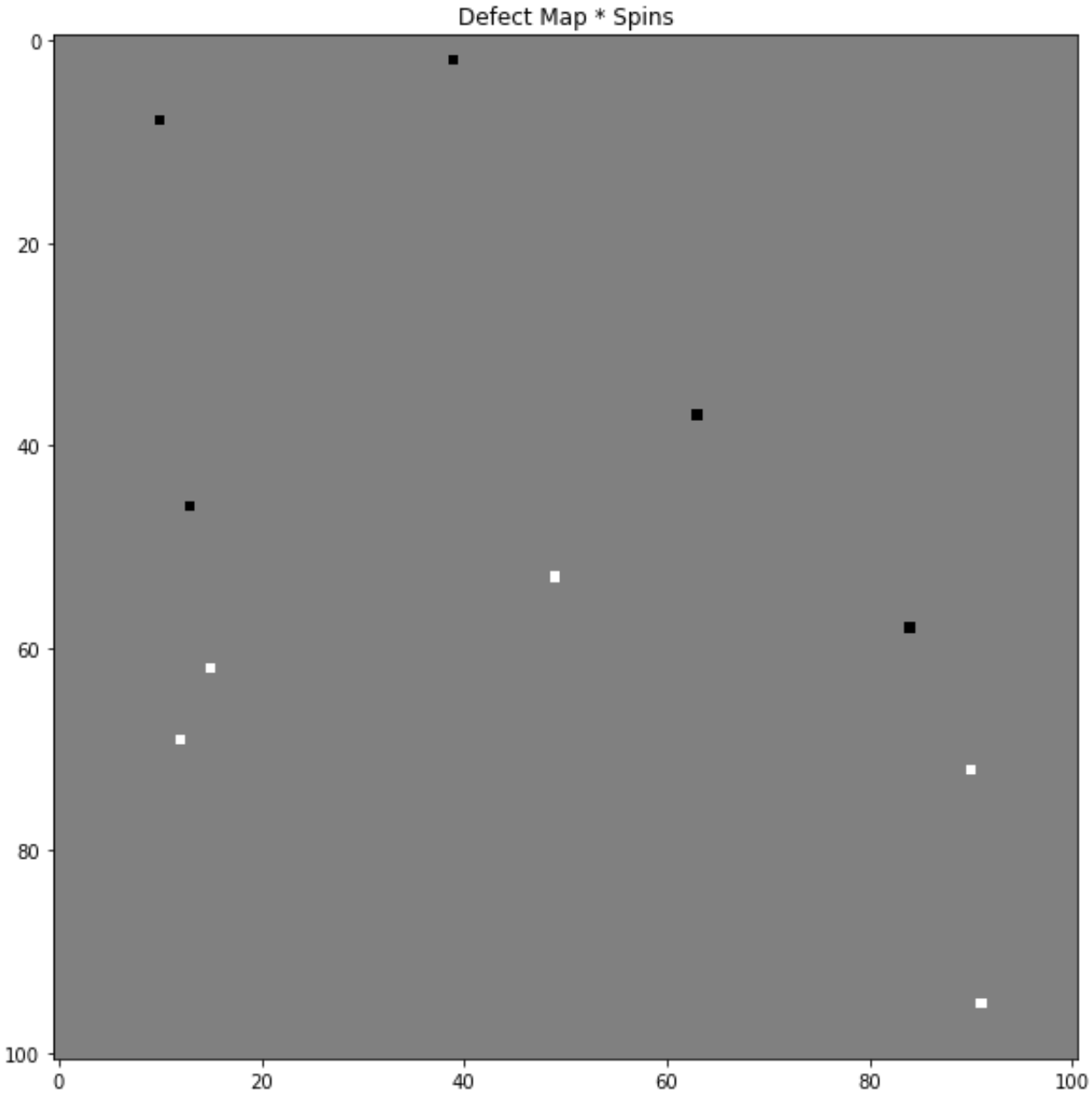
In [4]:

```
# Introduce defects in the lattice
Ndefects = 10  # Try 10 and 1000 for a nice contrast of possibilitie
(Nx,Ny) = spins.shape
defect_map = ( np.zeros(spins.shape, dtype='uint8') == 1 ) # Array o
x = np.random.randint(0,Nx,size=(Ndefects))
y = np.random.randint(0,Ny,size=(Ndefects))
defect_map[x,y]=True
#spins[x,y]=0
```

In [5]:
```
1 plt.figure(figsize=(15,10))
2 plt.imshow(defect_map,cmap='gray')
3 plt.title('Defect Map')
4 plt.show()
```

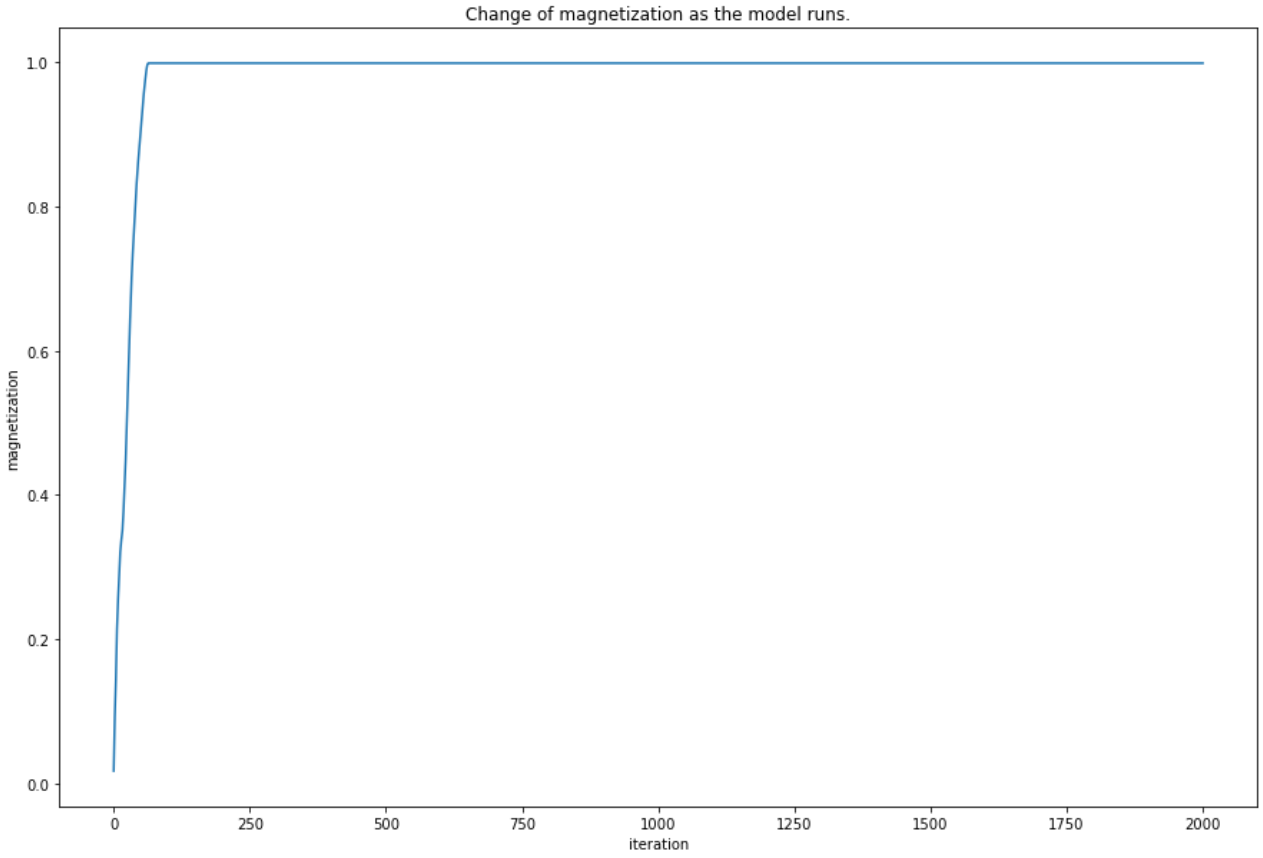


In [6]:
```
plt.figure(figsize=(15,10))
plt.imshow(defect_map * spins,cmap='gray')
plt.title('Defect Map * Spins')
plt.show()
```

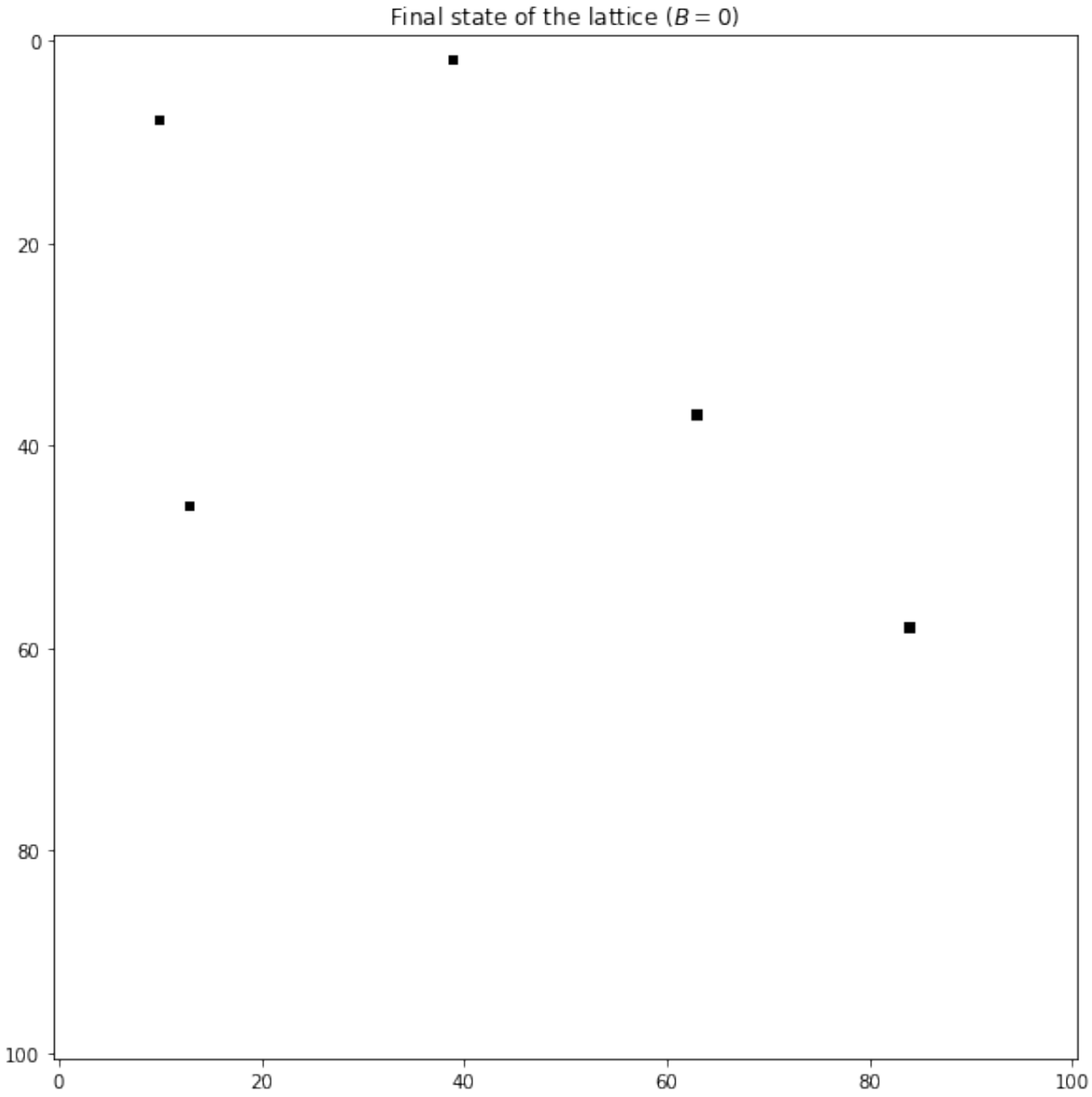


In [7]:
```
# Allow the model to evolve
kTamb = 0.5  # Ambient temperature
Bapplied = 0.0
Nwait = 2000

M = fe.many_smart_iterations(spins, Bapplied, kTamb, Nwait, defects=
```

In [8]:
```
# Did the magnetization change?
plt.figure(figsize=(15,10))
plt.plot(M)
plt.xlabel('iteration')
plt.ylabel('magnetization')
plt.title('Change of magnetization as the model runs.')
plt.show()
```



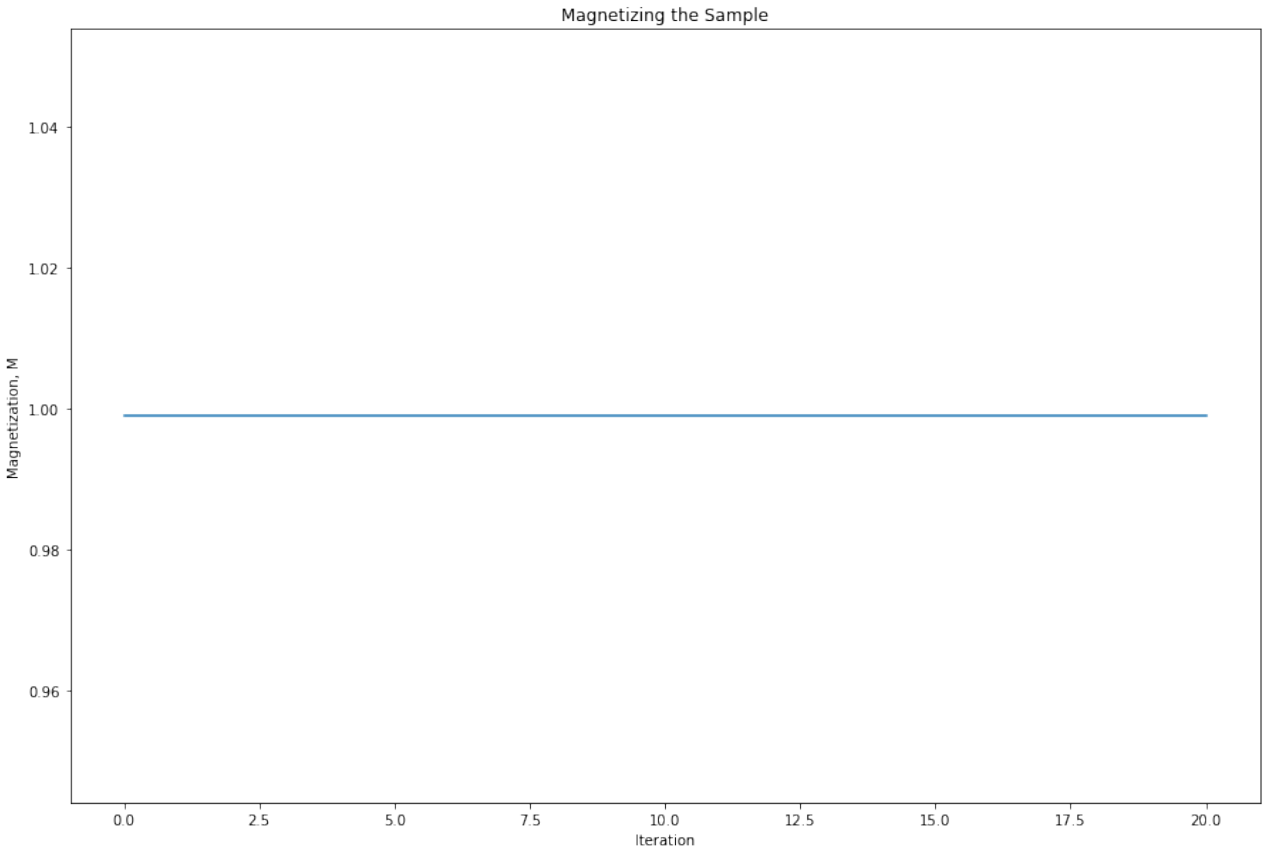
In [9]:
```
plt.figure(figsize=(15,10))
plt.imshow(spins,cmap='gray')
plt.title('Final state of the lattice ($B=0$)')
plt.show()
print('Final magnetization = ', fe.magnetization(spins))
```



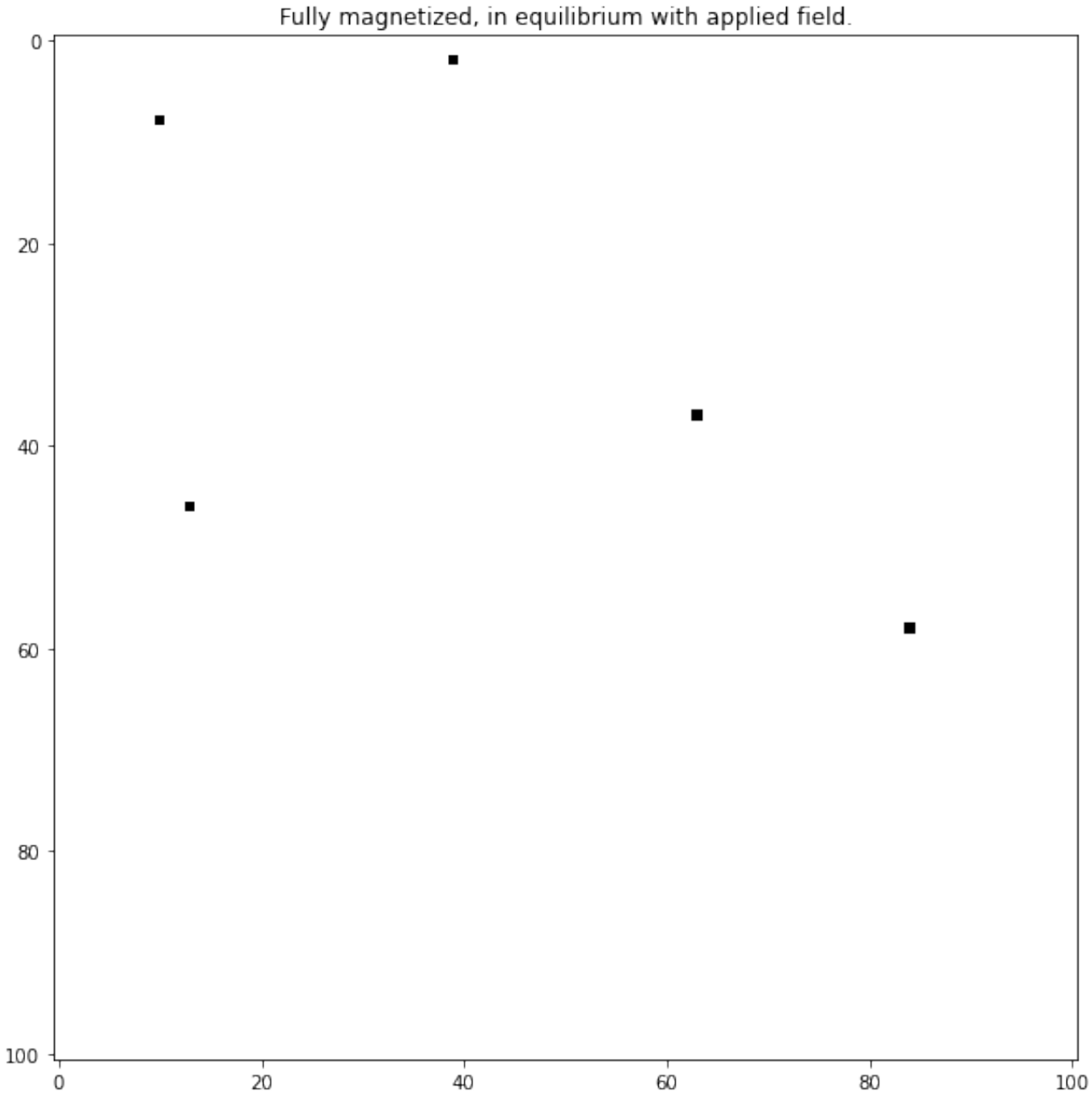

```
Final magnetization =  0.999019703950593
```

In [10]:
```
# Magnetize
Bmax = 2.0
Nwait = 20

M = fe.many_smart_iterations(spins, Bmax, kTamb, Nwait, defects=defe
```

In [11]:
```
plt.figure(figsize=(15,10))
plt.plot(M)
plt.xlabel('Iteration')
plt.ylabel('Magnetization, M')
plt.title('Magnetizing the Sample')
plt.show()
```



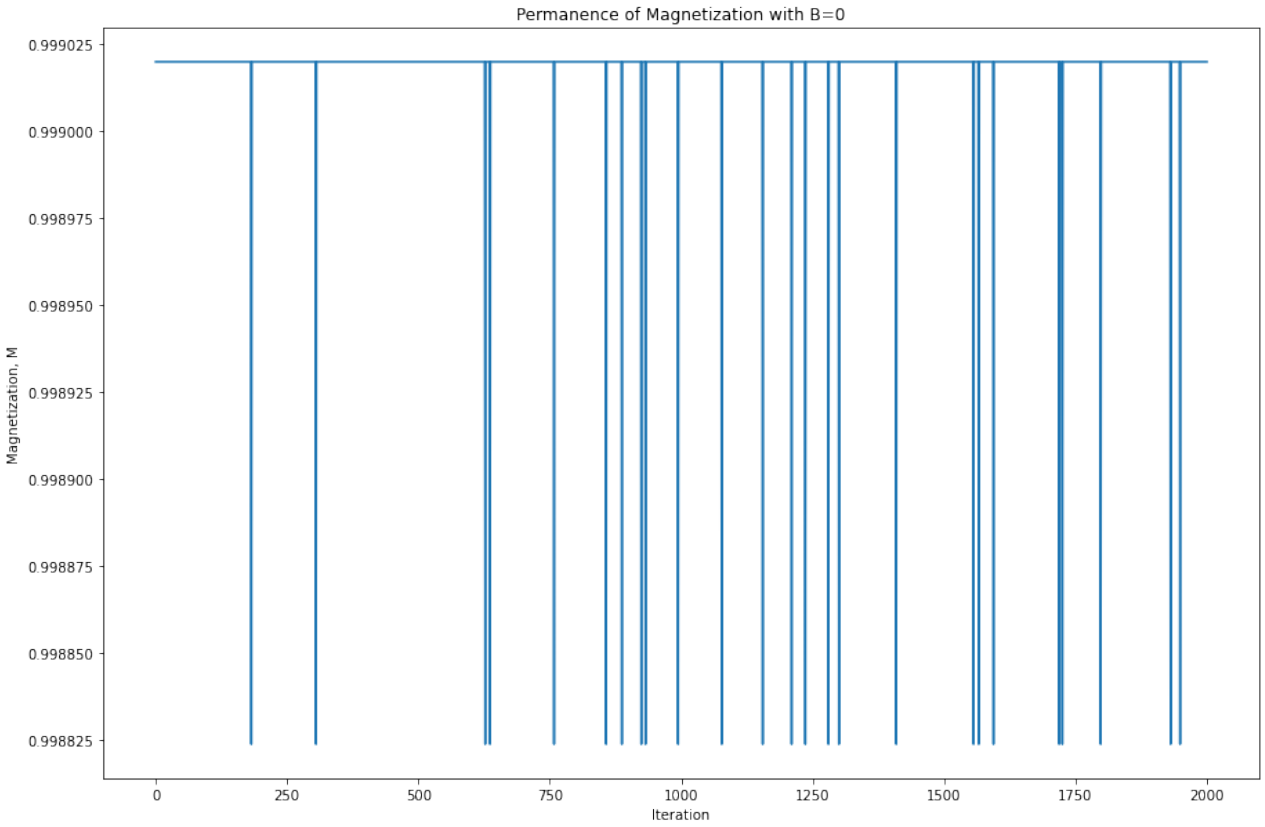
In [12]:
```
plt.figure(figsize=(15,10))
plt.imshow(spins,cmap='gray')
plt.title('Fully magnetized, in equilibrium with applied field.')
plt.show()
print('Final magnetization = ', M[-1])
```



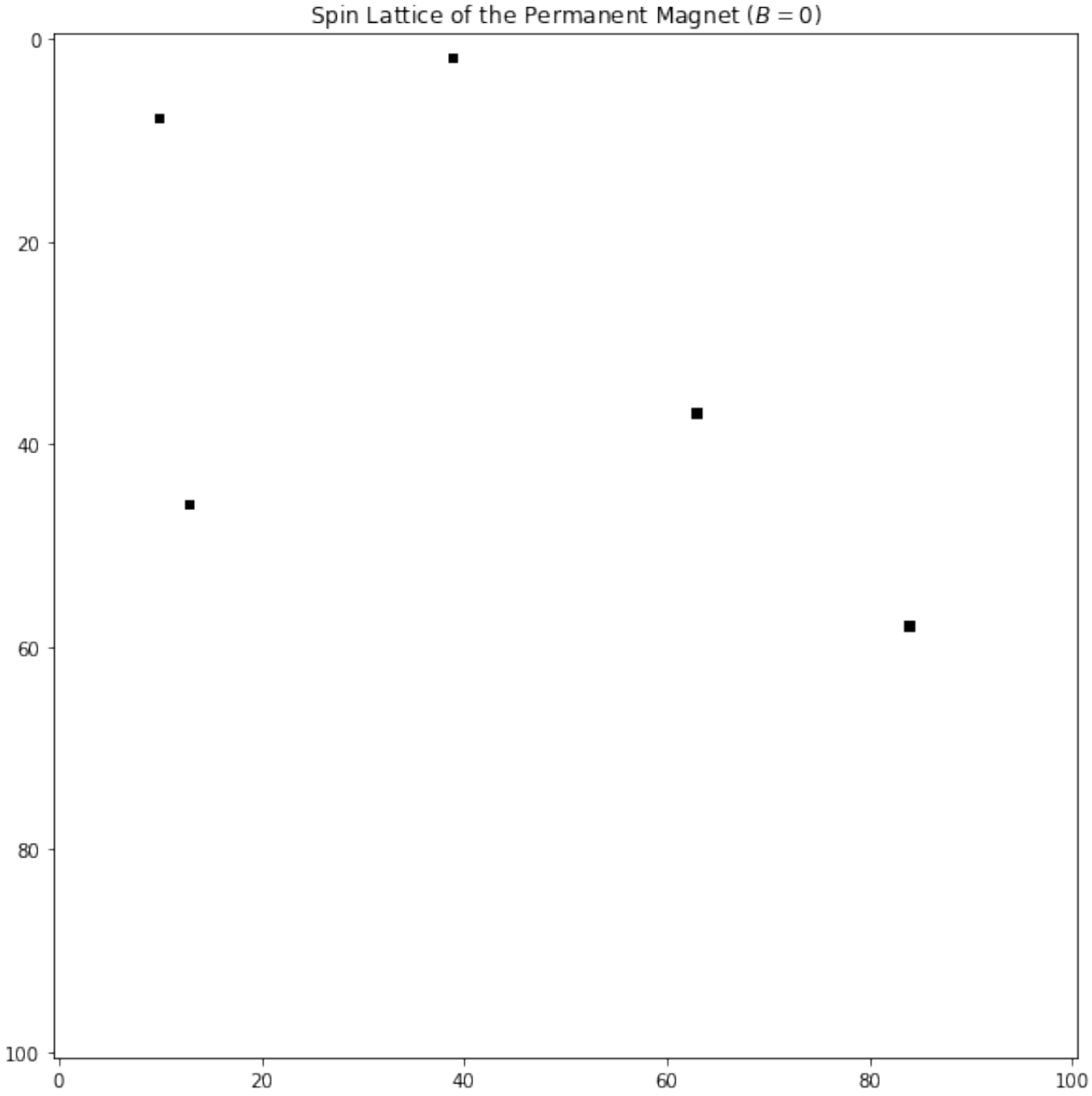

```
Final magnetization =  0.999019703950593
```

In [13]:
```
# Does it remain magnetized?
Bapplied = 0.0
Nwait = 2000

M = fe.many_smart_iterations(spins, Bapplied, kTamb, Nwait, defects=
```

In [14]:
```
plt.figure(figsize=(15,10))
plt.plot(M)
plt.xlabel('Iteration')
plt.ylabel('Magnetization, M')
plt.title('Permanence of Magnetization with B=0')
plt.show()
```



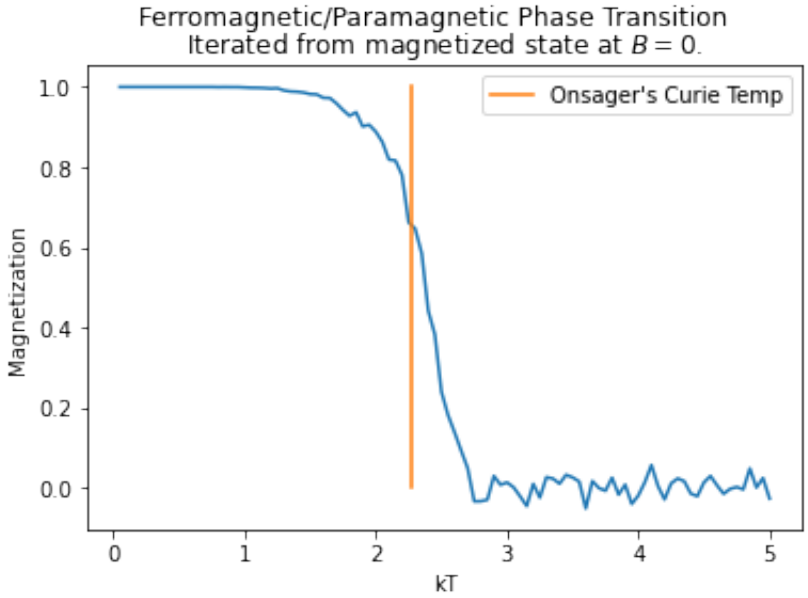
In [15]:
```
plt.figure(figsize=(15,10))
plt.imshow(spins,cmap='gray')
plt.title('Spin Lattice of the Permanent Magnet ($B=0$)')
plt.show()
print('Final magnetization = ', M[-1])
```




```
Final magnetization =  0.999019703950593
```

In [16]:
```
# Many runs from the magnetized state, at different temperatures
Nsteps = 100
kTmax = 5.0
kTramp = np.arange(1,Nsteps+1)*kTmax/Nsteps
Bapplied = 0.0
Mheating = np.empty((Nsteps))
for i in range(Nsteps):
    spins_copy = spins.copy() # For each kT step, start from the sam
    fe.many_smart_iterations(spins_copy, Bapplied, kTramp[i], 20, de
    Mheating[i] = fe.magnetization(spins_copy)

```

In [17]:
```
plt.plot(kTramp, Mheating)
plt.plot( np.array((2.269,2.269)), np.array((0,1)), label="Onsager's
plt.xlabel('kT')
plt.ylabel('Magnetization')
plt.suptitle('Ferromagnetic/Paramagnetic Phase Transition')
plt.title('Iterated from magnetized state at $B=0$.')
plt.legend()
plt.show()
print("Note that Onsager's analytic calculation did not include defe
```

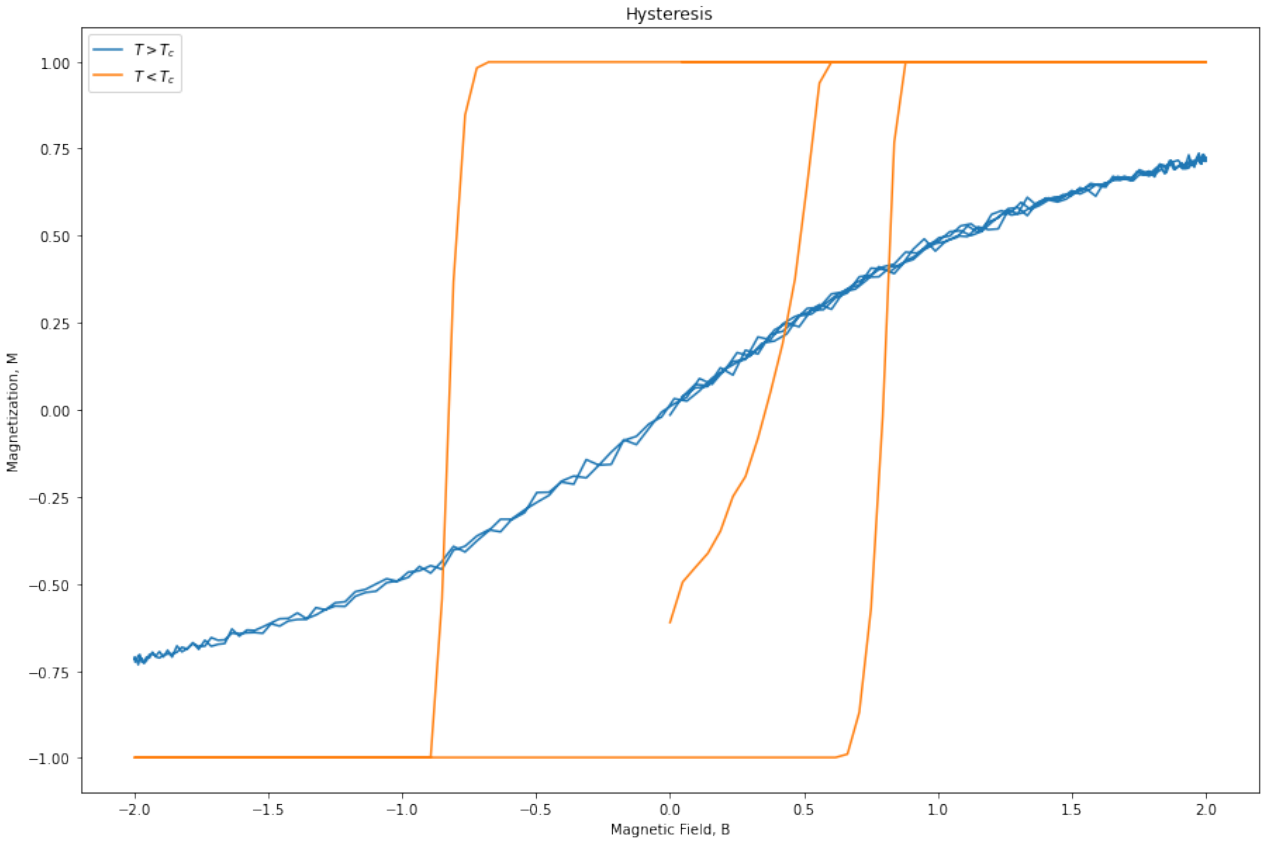



```
Note that Onsager's analytic calculation did not include defects.
```

In [18]:
```
# Hysteresis Curve at High Temperature
spins = spins_copy.copy() # Start out demagnetized.
Nsteps = 400
Bhot = np.empty((Nsteps))
Mhot = np.empty((Nsteps))
for i in range(Nsteps):
    Bhot[i] = Bmax * np.sin(3*np.pi*i/Nsteps)
    fe.many_smart_iterations(spins, Bhot[i], kTmax, 50, defects=defe
    Mhot[i] = fe.magnetization(spins)
```

In [19]:
```
# Hysteresis Curve at Ambient Temperature
spins = spins_copy.copy() # Start out demagnetized.
Nsteps = 400
B = np.empty((Nsteps))
M = np.empty((Nsteps))
for i in range(Nsteps):
    B[i] = Bmax * np.sin(3*np.pi*i/Nsteps)
    fe.many_smart_iterations(spins, B[i], kTamb, 50, defects=defect_
    M[i] = fe.magnetization(spins)
```

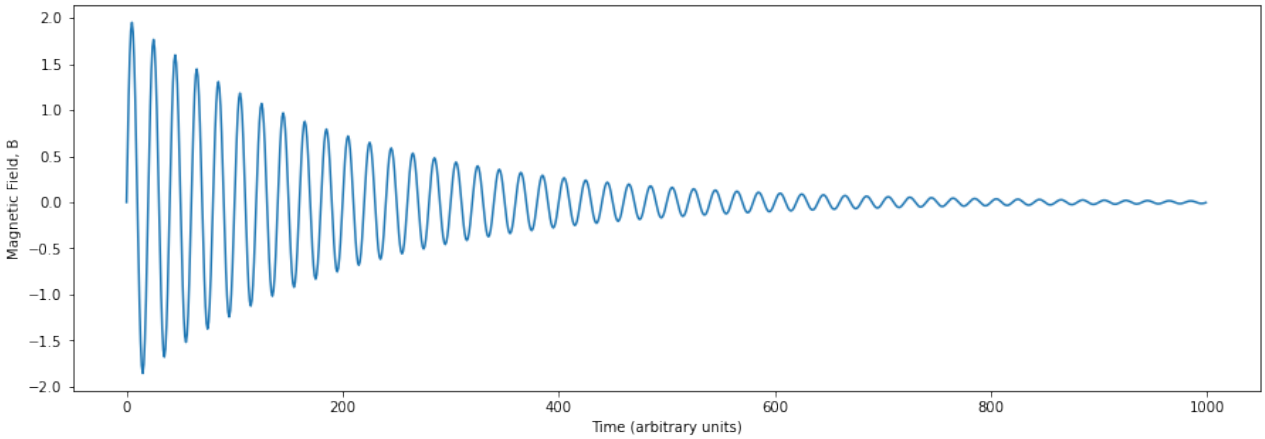
In [20]:
```
plt.figure(figsize=(15,10))
plt.plot(Bhot, Mhot, label='$T>T_c$')
plt.plot(B, M,       label='$T<T_c$')
plt.xlabel('Magnetic Field, B')
plt.ylabel('Magnetization, M')
plt.title('Hysteresis')
plt.legend()
plt.show()
```



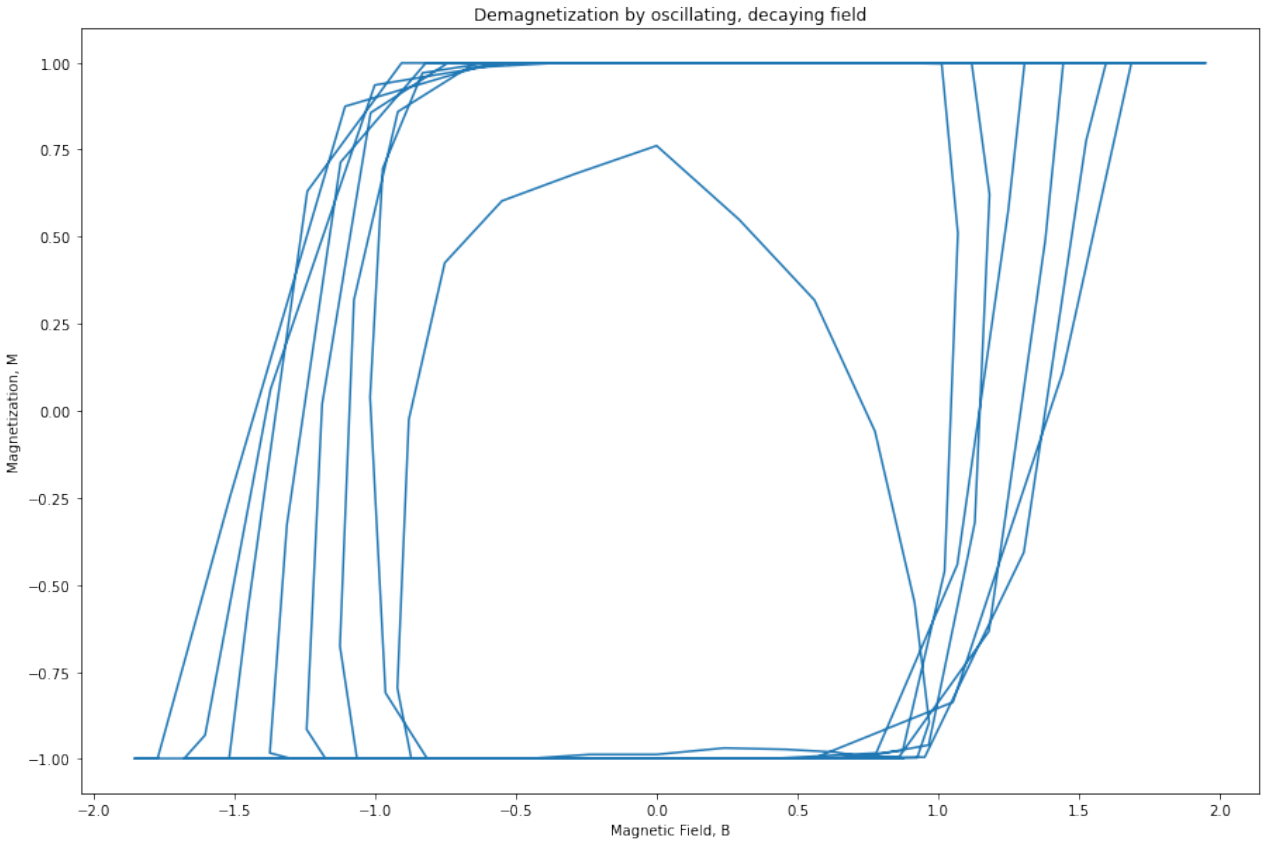
In [21]:
```
# Demagnetize
Nsteps = 1000
period = 20
decaytime = 10*period
B = np.empty((Nsteps))
M = np.empty((Nsteps))
for i in range(Nsteps):
    B[i] = Bmax * np.exp(-i/decaytime) * np.sin(2*np.pi*i/period)
    fe.many_smart_iterations(spins, B[i], kTamb, 20, defects=defect_
    M[i] = fe.magnetization(spins)
```

In [22]:
```
1 plt.figure(figsize=(15,5))
2 plt.plot(B)
3 plt.ylabel('Magnetic Field, B')
4 plt.xlabel('Time (arbitrary units)')
```

Out[22]: Text(0.5, 0, 'Time (arbitrary units)')

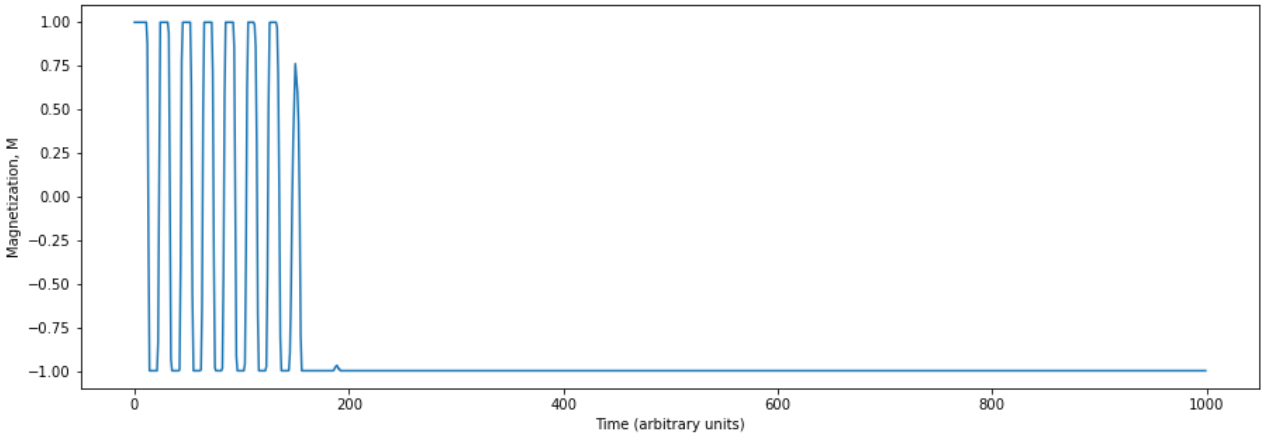


In [23]:
```
1 plt.figure(figsize=(15,10))
2 plt.plot(B,M)
3 plt.xlabel('Magnetic Field, B')
4 plt.ylabel('Magnetization, M')
5 plt.title('Demagnetization by oscillating, decaying field')
6 plt.show()
```

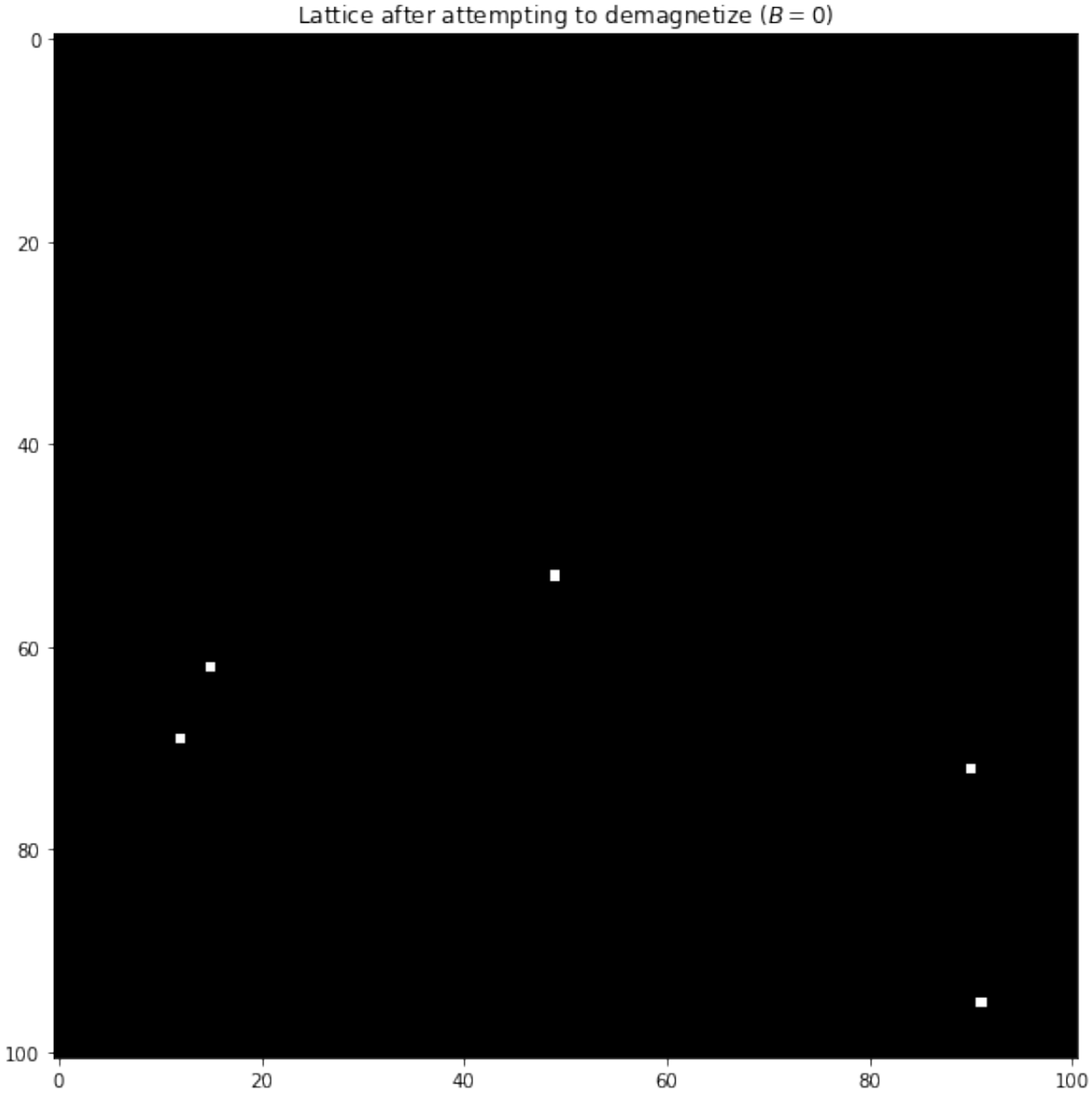


In [24]:
```
1 plt.figure(figsize=(15,5))
2 plt.plot(M)
3 plt.ylabel('Magnetization, M')
4 plt.xlabel('Time (arbitrary units)')
```

Out[24]: Text(0.5, 0, 'Time (arbitrary units)')



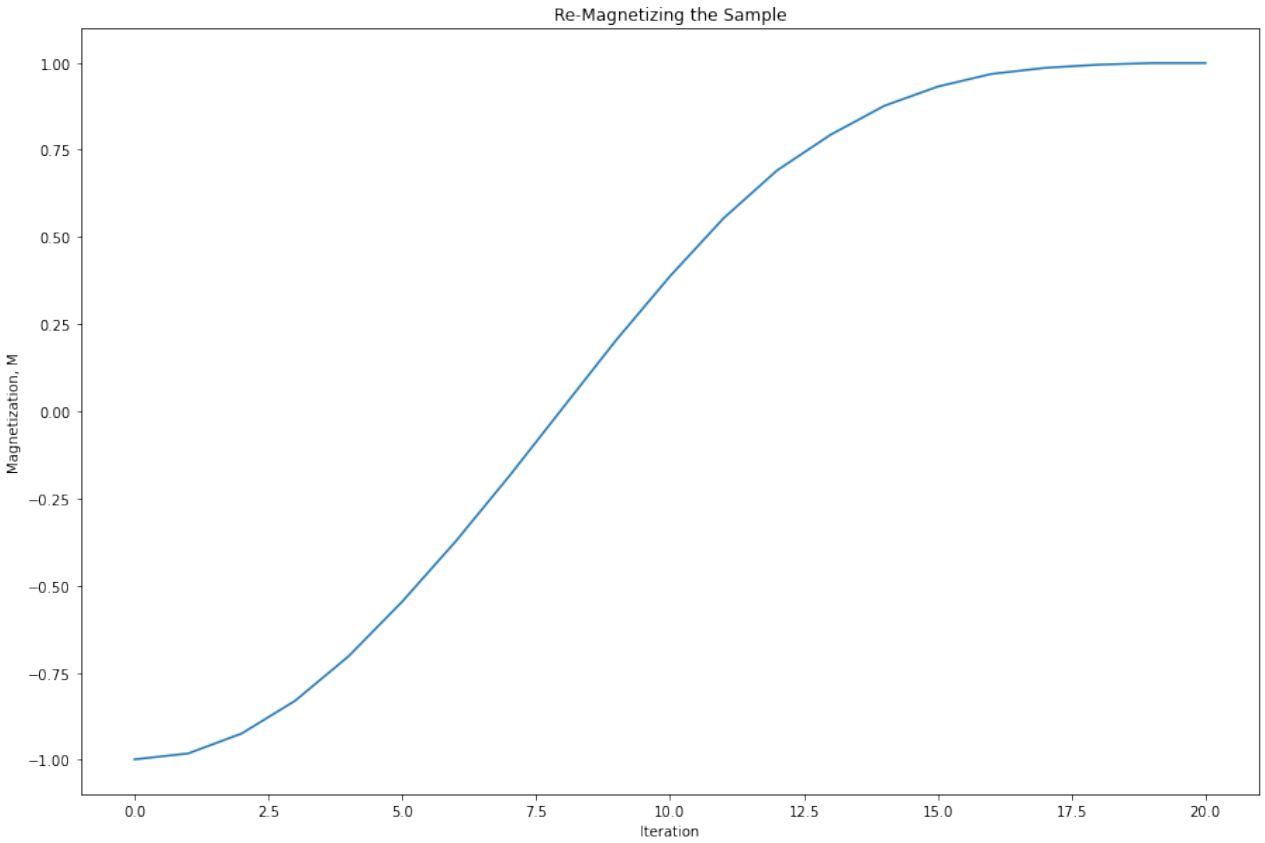
In [25]:
```
plt.figure(figsize=(15,10))
plt.imshow(spins,cmap='gray')
plt.title('Lattice after attempting to demagnetize ($B=0$)')
plt.show()
print('Final magnetization = ', fe.magnetization(spins))
```




```
Final magnetization =  -0.999019703950593
```

In [26]:
```
# Re-Magnetize
Bmax = 2.0
Nwait = 20

M = fe.many_smart_iterations(spins, Bmax, kTamb, Nwait, defects=defe
```

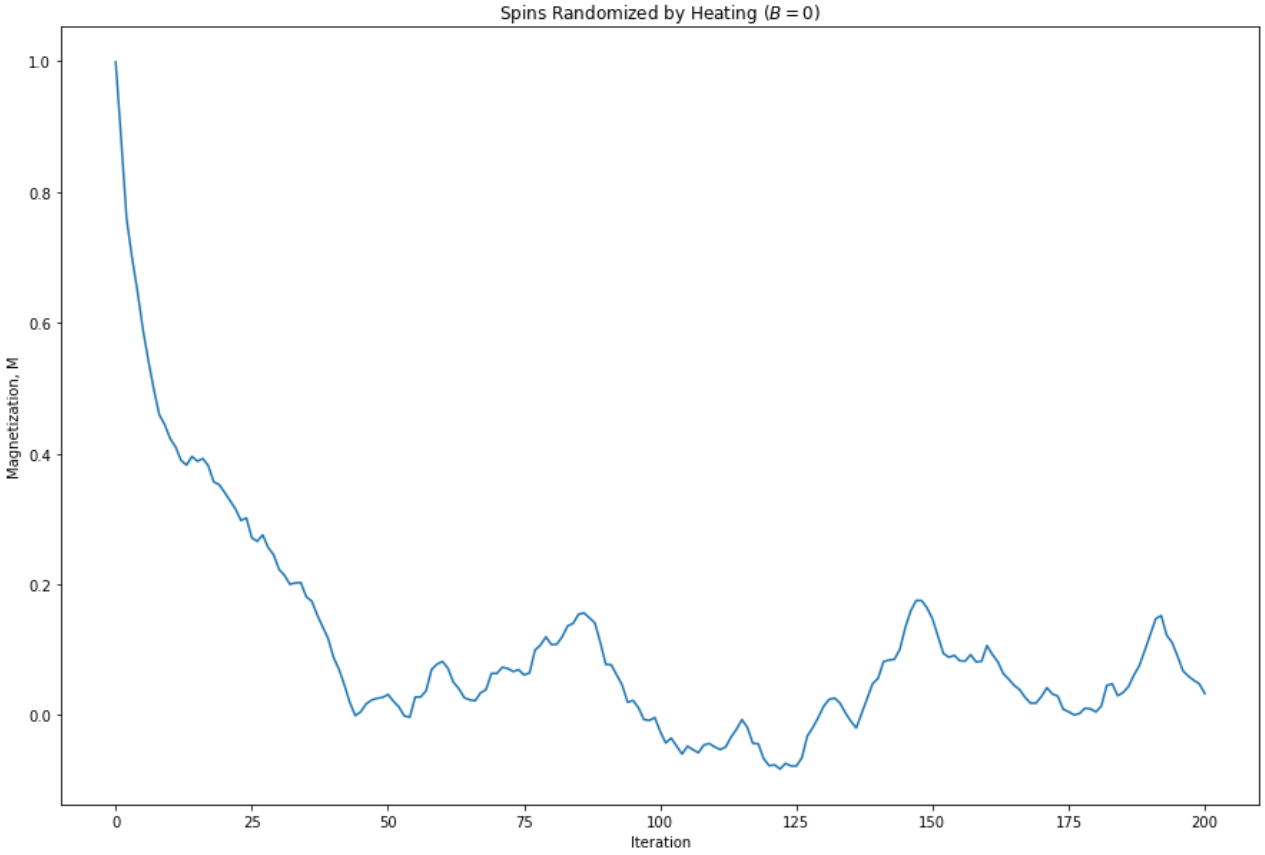
In [27]:
```
plt.figure(figsize=(15,10))
plt.plot(M)
plt.xlabel('Iteration')
plt.ylabel('Magnetization, M')
plt.title('Re-Magnetizing the Sample')
plt.show()
```



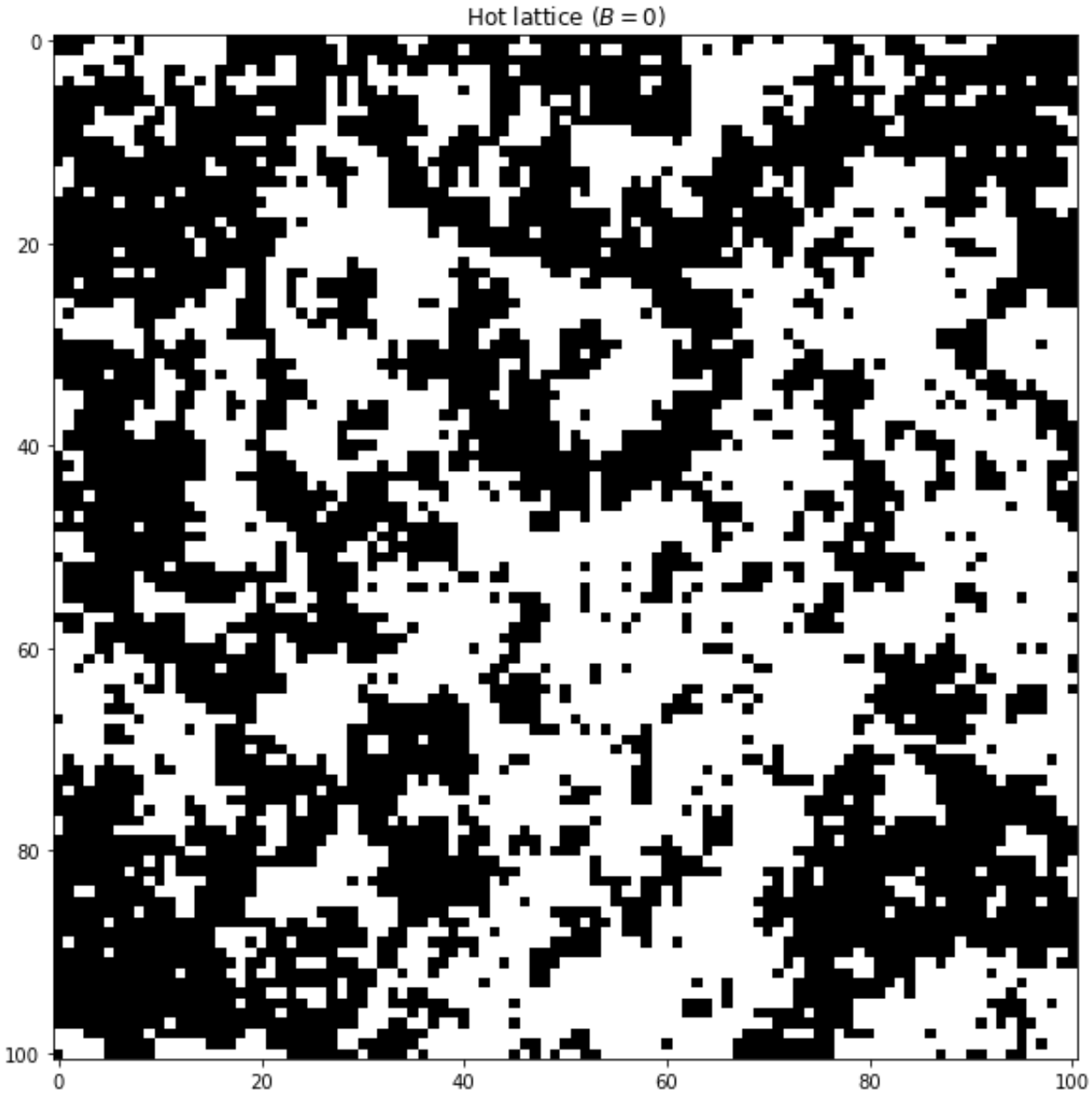
In [28]:
```
# Demagnetize by heating
kT_hot = 2.5
Bapplied=0
M = fe.many_smart_iterations(spins, Bapplied, kT_hot, 200)
spins_randomized = spins.copy() # save for future reference
```

In [29]:

```
plt.figure(figsize=(15,10))
plt.plot(M)
plt.xlabel('Iteration')
plt.ylabel('Magnetization, M')
plt.title('Spins Randomized by Heating ($B=0$)')
plt.show()
```



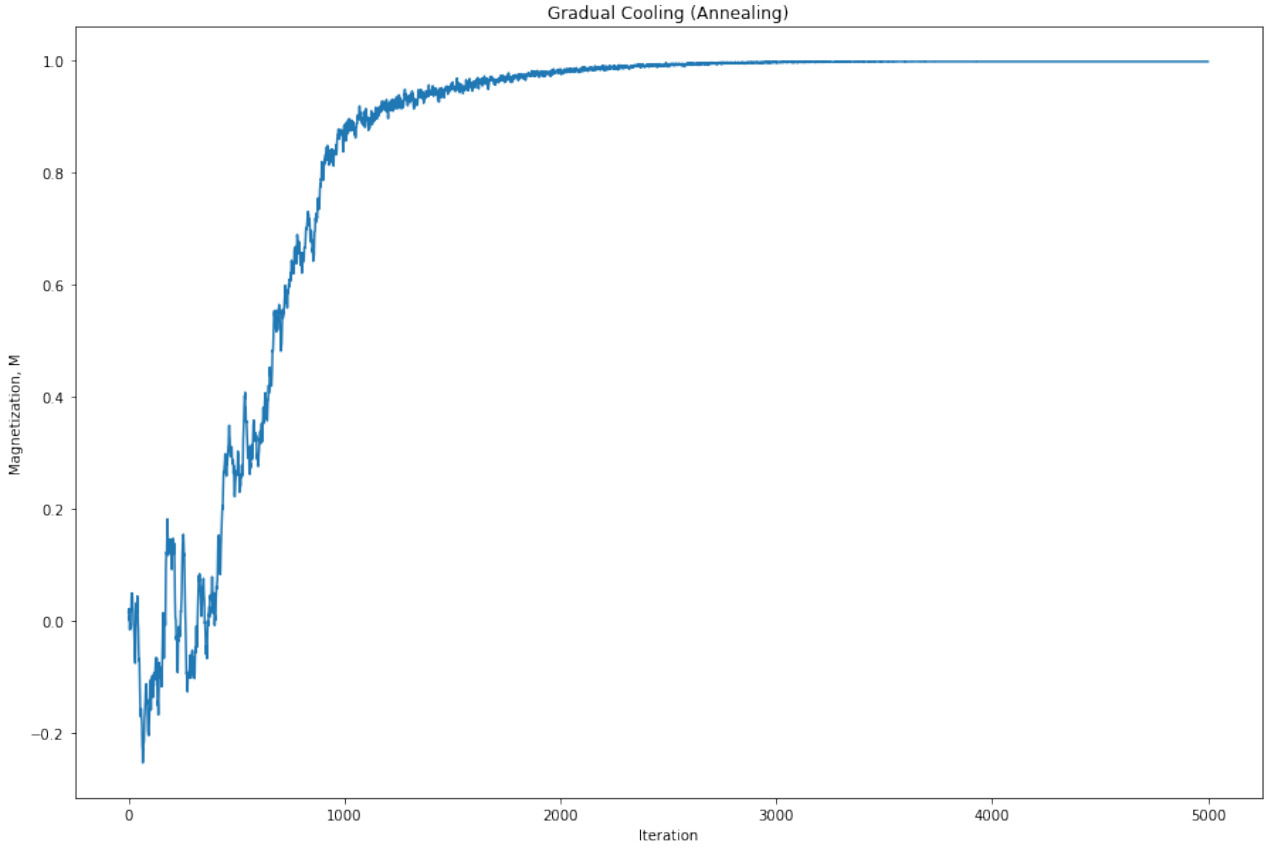
In [30]:
```
plt.figure(figsize=(15,10))
plt.imshow(spins,cmap='gray')
plt.title('Hot lattice ($B=0$)')
plt.show()
print('Magnetization = ', fe.magnetization(spins))
```



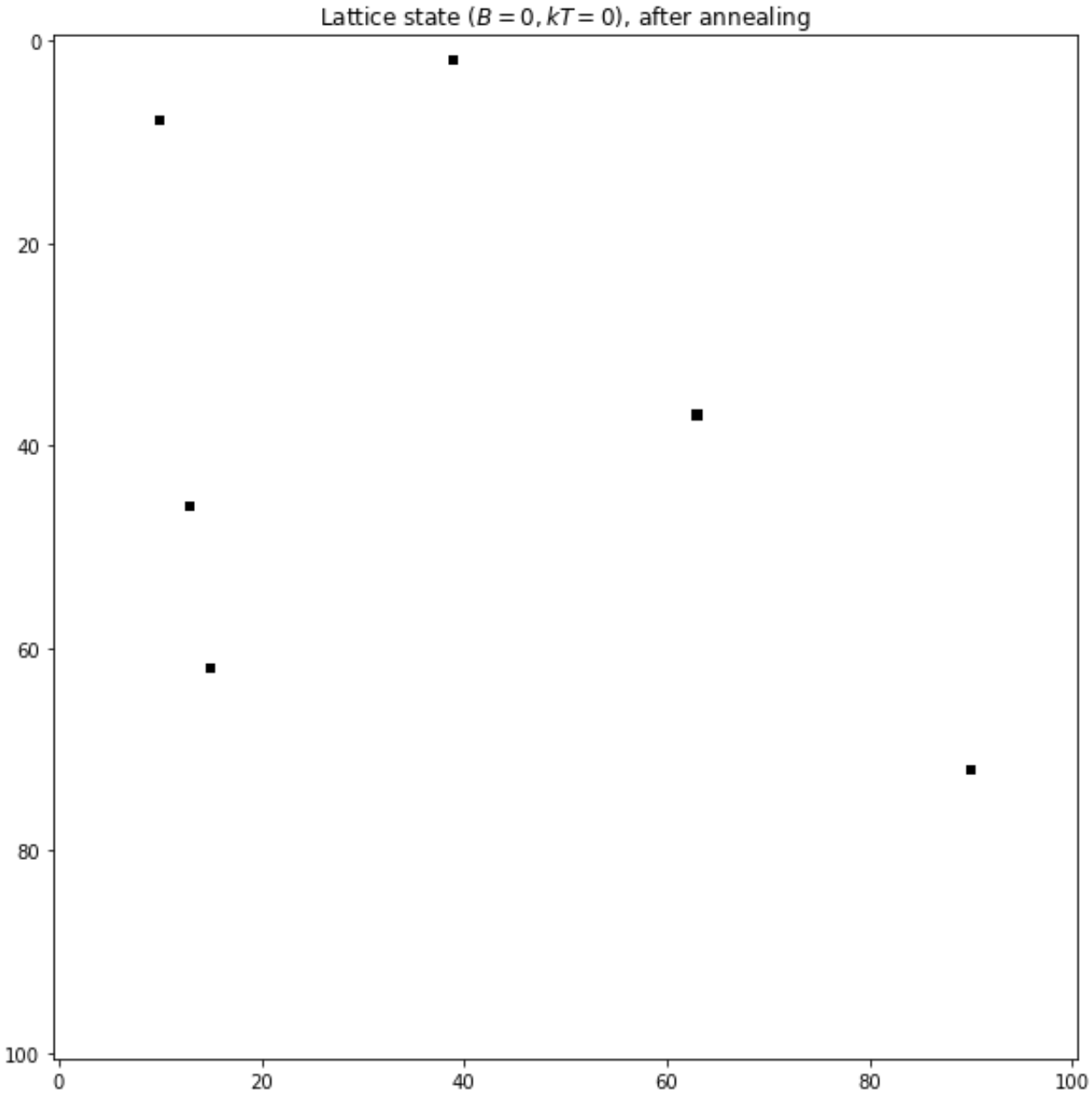

```
Magnetization =  0.032447799235369085
```

In [31]:
```
# Cool gradually, see what happens to magnetization.
spins = spins_randomized.copy()  # Start out with spins randomized by
deltaT = 0.99*kT_hot
Ncool = 5000
kT_range = kT_hot - deltaT * np.arange(Ncool)/Ncool
Bapplied=0.0
M = np.empty((Ncool))
for i in range(Ncool):
    fe.smart_iterate(spins, Bapplied, kT_range[i], defects=defect_map
    M[i] = fe.magnetization(spins)
```

In [32]:
```
plt.figure(figsize=(15,10))
plt.plot(M)
plt.xlabel('Iteration')
plt.ylabel('Magnetization, M')
plt.title('Gradual Cooling (Annealing)')
plt.show()
```



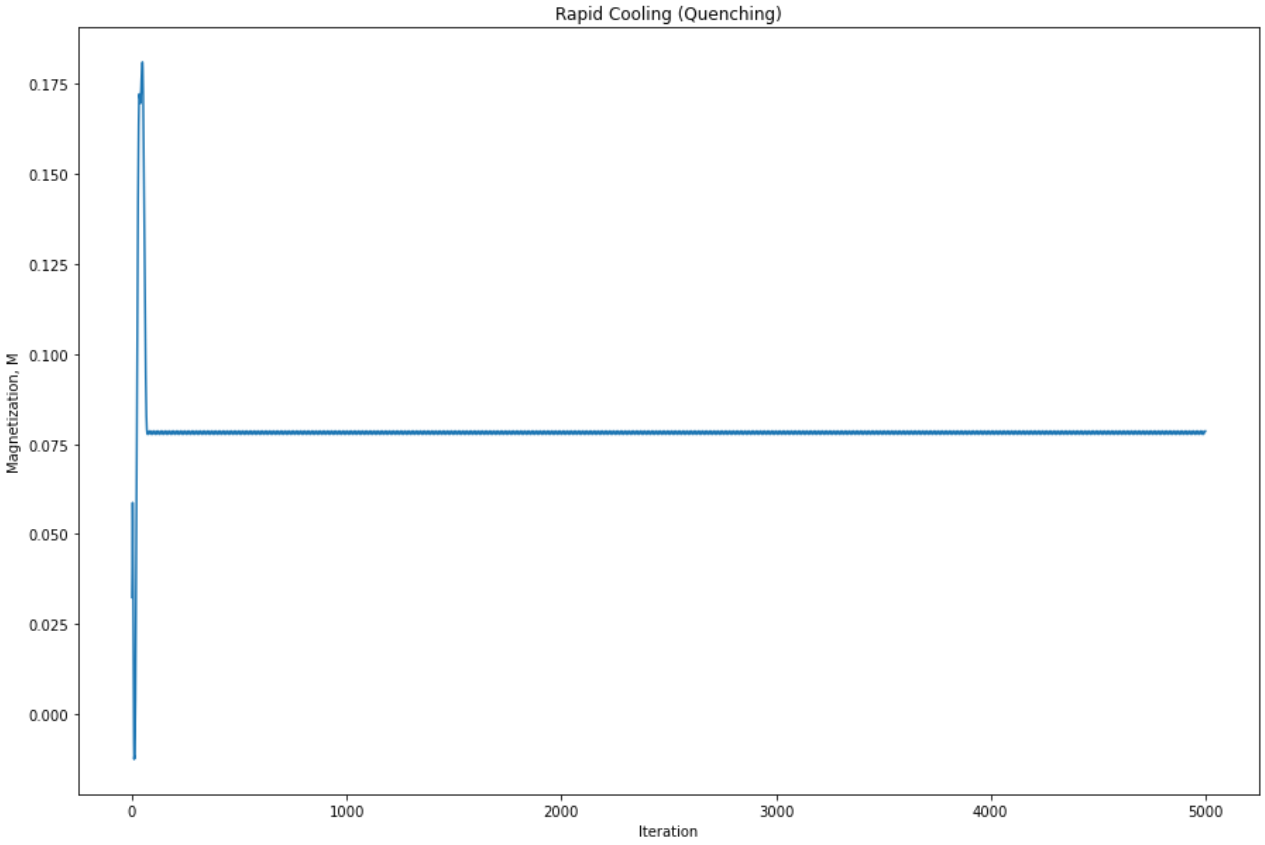
In [33]:
```
plt.figure(figsize=(15,10))
plt.imshow(spins,cmap='gray')
plt.title('Lattice state ($B=0, kT = 0$), after annealing')
plt.show()
print('Final magnetization = ', fe.magnetization(spins))
```



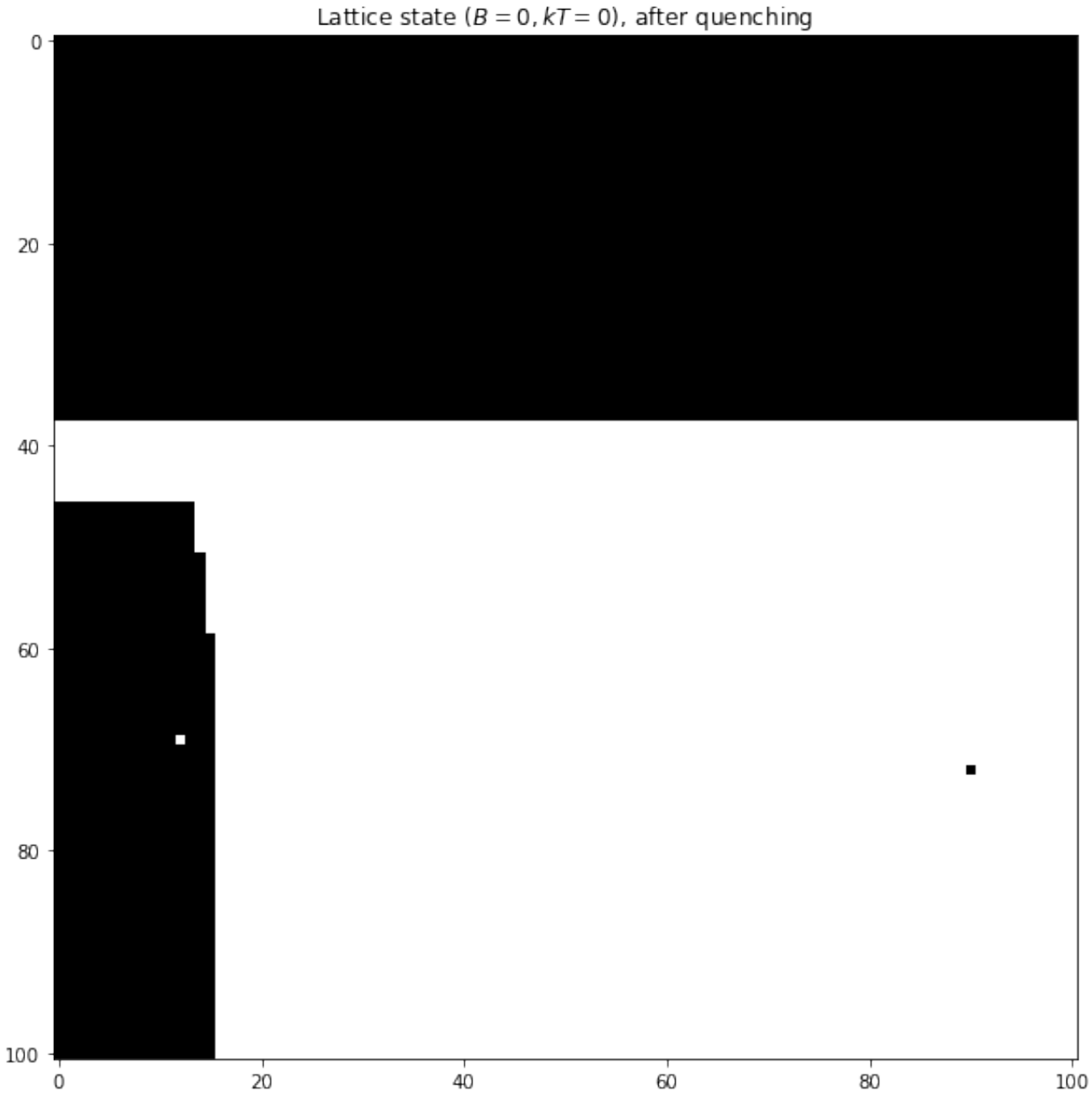

```
Final magnetization =  0.9988236447407117
```

In [34]:
```
# Quench
kT_cool = 0.01
spins = spins_randomized.copy()  # Start out with spins randomized by

M = fe.many_smart_iterations(spins, Bapplied, kT_range[i], 5000, defe

```

In [35]:
```
plt.figure(figsize=(15,10))
plt.plot(M)
plt.xlabel('Iteration')
plt.ylabel('Magnetization, M')
plt.title('Rapid Cooling (Quenching)')
plt.show()
```



In [36]:
```
plt.figure(figsize=(15,10))
plt.imshow(spins,cmap='gray')
plt.title('Lattice state ($B=0, kT=0$), after quenching')
plt.show()
print('Final magnetization = ', fe.magnetization(spins))
```




```
Final magnetization =  0.07852171355749436
```