Here are the detailed solutions to the questions provided.

---

1. Dynamical System in Neurodynamics

In its essence, a **dynamical system** is any system whose state evolves over time according to a fixed mathematical rule. In the specific context of **neurodynamics**, this concept is used to model the brain and its components.

Here, the "state" of the system represents the changing properties of neurons. This could be:

- The **membrane potential** (voltage) of a single neuron.
- The collective **firing rates** of a population of neurons.
- The status (open or closed) of various **ion channels** in a neuron's membrane.

The "rule" that governs the system's evolution is typically a set of **differential equations** (for continuous changes over time) or **difference equations** (for discrete time steps). These equations describe how neurons influence one another and respond to stimuli.

For example, a very simple model for a single neuron's voltage (V) might be dtdV=−V+I, where I is an input current. This equation is the rule, telling us that the voltage tends to decay back to zero but is pushed up by the input current. A neurodynamical system, then, is a mathematical framework for describing the complex, time-varying dance of neural activity.

---

2. Stability of Equilibrium States

An **equilibrium state**, or fixed point, is a state of a system where all change has ceased—it is perfectly balanced and will remain there forever if undisturbed. Think of a pendulum that has come to a complete stop, hanging straight down. The **stability** of this equilibrium describes what happens when the system is given a small nudge.

We can visualize this with a simple analogy:

- **Stable Equilibrium**: Imagine a ball resting at the bottom of a valley. If you push the ball slightly, it will roll back down and settle at the bottom again. This is a stable state.
- **Unstable Equilibrium**: Imagine a ball balanced perfectly on top of a hill. The slightest puff of wind will cause it to roll down the side, moving far away from its original position. This is an unstable state.

To determine stability mathematically, two primary methods are used:

1. **Linearization Method**: This powerful technique involves examining the system's behavior in the immediate vicinity of an equilibrium point, where it can be accurately approximated by a simpler linear system. The process involves calculating the **Jacobian matrix** (the multi-dimensional equivalent of a derivative) at the equilibrium point. The **eigenvalues (λ)** of this matrix hold the key:

    - For continuous-time systems, if all eigenvalues have **negative real parts**, any small perturbation will decay exponentially, and the system will return to equilibrium. The point is **stable**. If even one eigenvalue has a positive real part, a perturbation along that direction will grow exponentially, making the point **unstable**.
2. **Lyapunov's Method**: This method takes a more global, energy-based approach. The goal is to find a special function called a **Lyapunov function** (V(x)), which acts like an "energy" measure for the system. An equilibrium point is proven to be stable if we can find a V(x) that satisfies two conditions:
    - It is positive for all nearby states and is zero only at the equilibrium point itself (like how potential energy is lowest at the bottom of a valley).
    - Its time derivative (dtdV) is always negative or zero. This means the system's "energy" is always decreasing as it moves, so it must be heading towards the lowest energy state—the stable equilibrium.

---

3. Attractors and Their Function

Attractors are the states or patterns of behavior that a dynamical system naturally settles into over time. An attractor can be a single point, a repeating loop, or a more complex pattern. The set of all starting conditions that lead to a particular attractor is known as its **basin of attraction**.

Think of a landscape with several valleys. Each valley floor is an attractor. No matter where you release a ball on the slope of a particular valley, it will eventually end up on that valley's floor. The entire slope leading to that valley is its basin of attraction.

In neurodynamics, attractors are profoundly important because they represent the **stable, robust outputs or memories of the brain**. Their function is to give the system predictable long-term behaviors.

- **Memory and Pattern Completion**: A stored memory can be modeled as an attractor. The brain doesn't need a perfect cue to recall something; a partial or "noisy" cue (an initial state within the basin of attraction) is enough. The network's dynamics will automatically pull the state towards the complete, correct memory (the attractor). This makes memory robust and error-tolerant.
- **Decision Making**: Different choices can be represented by different attractors. As evidence accumulates, the system's state is pushed around until it crosses into one basin of attraction, locking in the corresponding decision.
- **Motor Control**: Rhythmic behaviors like walking are governed by **limit cycle attractors**, where the neural activity follows a stable, repeating loop, generating a consistent motor output.

---

4. Fixed Point vs. Limit Cycle Attractors

Fixed Point Attractors and **Limit Cycle Attractors** are two fundamental types of attractors that represent distinct forms of stable behavior in a system.

Feature	Fixed Point Attractor	Limit Cycle Attractor
Description	A single, stationary point in the system's state space. It represents a state of perfect, unchanging equilibrium.	A closed, isolated trajectory (a loop) in the state space. It represents a perfectly stable, periodic oscillation.
System Behavior	The system converges towards this single point and then becomes **static**, remaining fixed for all time.	The system converges towards this loop and then enters a state of **perpetual oscillation**, cycling through the same sequence of states repeatedly and predictably.
Neurodynamic Example	A **Hopfield network** settling on a stored memory. The final pattern of neuron activations is static and stable—a fixed point.	A **Central Pattern Generator (CPG)**, the neural circuit in the spinal cord that produces the rhythmic muscle commands for walking. The firing pattern of the neurons in the CPG is a stable, repeating cycle.

---

5. Manipulation of Attractors in Recurrent Networks

Recurrent neural networks, with their looping, feedback connections, are natural dynamical systems. The process of "training" or "programming" such a network is fundamentally about **manipulating its attractors**.

Imagine the network's possible states as a vast landscape. Training is the art of **sculpting this landscape**. The steps are:

1. **Define Desired Memories**: First, we identify the specific patterns of neural activity that we want the network to "remember" or treat as stable outputs.

2. **Adjust Connection Weights**: The primary tool for sculpting is the adjustment of the **synaptic weights** between neurons. Changing the weights is like using a chisel to alter the shape of the energy landscape.
3. **Create Attractor Basins**: By applying a learning rule (like the Hebbian rule), the weights are modified to create deep "valleys" (low-energy attractors) at the precise locations of the desired memory patterns. This process simultaneously carves out the surrounding slopes, which become the basins of attraction for those memories.

Once the landscape is sculpted, the network is ready. When presented with a noisy or incomplete cue (an initial state), the system's dynamics take over. The state will naturally "roll downhill" on the energy landscape until it settles at the bottom of the nearest valley, thereby recalling the complete, error-free memory associated with that attractor.

---

6. Hopfield Network Architecture and Comparison

A **Hopfield network** is a specific and foundational type of recurrent neural network known for its ability to function as an associative memory.

Architecture:

- It has a **single layer** of neurons, which can be thought of as both input and output units.
- The neurons are **fully interconnected**—every neuron is connected to every other neuron.
- The connection weights are **symmetric**, meaning the strength of the connection from neuron A to B is identical to the connection from B to A (WAB=WBA). This is a critical feature that allows for the existence of a global energy function.
- Neurons do not have **self-connections** (WAA=0).
- The neurons themselves are typically **binary**, holding a state of either +1 (firing) or -1 (not firing).

Difference from Feedforward Networks:

The distinction between a Hopfield network and a standard feedforward network is fundamental, reflecting a difference between dynamic evolution and static mapping.

Feature	Hopfield Network (Recurrent)	Feedforward Network
Connections & Flow	**Recurrent & Bidirectional**. Connections form loops,	**Acyclic & Unidirectional**. Information flows in one direction, like a river, from an

	allowing information to circulate and evolve over time.	input layer to an output layer without any loops.
Computation	**Dynamic Evolution to an Attractor**. A Hopfield network is given an initial state and then runs freely, updating its neurons until it settles into a stable low-energy state (an attractor). It's a journey to a destination.	**Static Input-Output Mapping**. A feedforward network performs a one-shot calculation. An input is fed in, passes through the layers once, and a final output is produced. It's a direct transformation.
Purpose	**Associative Memory & Optimization**. Ideal for tasks like pattern completion, denoising, and solving certain optimization problems.	**Classification & Regression**. Ideal for tasks where you need to map a given input to a specific output label or value.

---

7. The Energy Function in Hopfield Models

The **energy function** is the theoretical cornerstone of the Hopfield network, explaining how it reliably finds stable states. For a network with neuron states Si and connection weights Wij, the energy (E) of any given configuration is defined as:

E=−21i=j∑WijSiSj
This equation essentially measures the total "conflict" or "tension" in the network. A lower energy corresponds to a more consistent or harmonious state. For example, if two neurons *i* and *j* are connected by a strong positive weight (Wij>0), the energy is lowest when they have the same state (SiSj=1). If they are connected by a negative (inhibitory) weight (Wij<0), the energy is lowest when they have opposite states (SiSj=−1).

How it drives the network to an attractor:

The network's update rule is ingeniously designed to always follow the path of steepest descent on this energy landscape. When a single neuron is updated asynchronously, it flips its state only if doing so **lowers the total energy of the network**.

1. **Guaranteed Descent**: Because every step either decreases the energy or leaves it unchanged, the network cannot get stuck in a loop. It must always proceed "downhill."

2. **Attractors as Valleys**: The stable states that the network can remember—the attractors—are precisely the **local minima** of this energy function. They are the bottoms of the valleys on the landscape.

Therefore, the network's dynamic evolution is not random; it is a deterministic process of energy minimization. By simply following its local update rule, the network as a whole is guaranteed to converge and settle into the nearest low-energy attractor state.

---

8. Atypical Attractors and Memory Capacity

A standard Hopfield network, trained with the simple Hebbian rule ("neurons that fire together, wire together"), has a surprisingly low **memory capacity**. It can only reliably store about 0.14 patterns for every N neurons in the network. If you try to store more, the "cross-talk" between memories creates **spurious attractors**—unintended stable states that are jumbled mixtures of the true memories, corrupting recall.

Improving this capacity requires moving beyond simple Hebbian learning to sculpt the energy landscape in more sophisticated, "atypical" ways.

1. **Unlearning**: This technique is like targeted maintenance for the energy landscape. The network is allowed to run and fall into a spurious attractor. Then, a small amount of "anti-Hebbian" learning is applied to that spurious state. This has the effect of "filling in the pothole"—it slightly raises the energy of that unintended valley, making it less likely to trap the network's state. Repeated unlearning smooths out the landscape, making the true memory attractors deeper and their basins of attraction wider and more dominant.
2. **Projection Methods (e.g., Gardner's Algorithm)**: These are far more advanced techniques. Instead of building up the weights pattern by pattern (which causes the cross-talk), they treat the problem as a constraint satisfaction problem. They ask: "What is the optimal weight matrix W that would make all of our desired patterns perfectly stable attractors?" These algorithms can design a highly complex, non-standard energy landscape from scratch, carefully engineered to contain only the desired attractors. Using such methods, the storage capacity can be dramatically increased to as high as $2N$ patterns.

---

9. Energy Minimization for Optimization Problems

The Hopfield network's innate ability to minimize its energy function can be cleverly harnessed to solve difficult **combinatorial optimization problems**. The core idea is to frame the optimization problem in the language of the network, turning the network into a specialized analog computer.

The process involves two key steps:

1. **Map Problem to Network**: The variables of the optimization problem are mapped to the states of the neurons, and most importantly, the problem's **cost function** (the quantity to be minimized) is encoded into the network's **energy function** by carefully designing the connection weights (Wij).
2. **Let the Network Find the Solution**: The network is initialized in a random state and allowed to run. Its natural physical dynamics will cause it to evolve towards a state of low energy. Because the energy function *is* the cost function, this low-energy state corresponds to a low-cost (and hopefully optimal or near-optimal) solution to the original problem.

Example Formulation: Traveling Salesman Problem (TSP)

- **Objective**: For *N* cities, find the shortest possible tour that visits each city exactly once.
- **Hopfield Formulation**:
    - **Neurons**: We use an *N* x *N* grid of neurons, where Sxi=1 means "city *x* is the *i*-th stop on the tour."
    - Energy Function: The cost function is translated into an energy function with several "penalty" terms and one "objective" term. The constants A, B, C, and D are used to weight the importance of each rule. E=2Ax∑i∑j=i∑SxiSxj+2Bi∑x∑y=x∑SxiSyi+2C(x∑i∑Sxi−N)2+2Dx∑y∑i∑dxySxi(Sy, i+1)
        - The **A term** is a penalty that is high if any city appears in more than one position of the tour.
        - The **B term** is a penalty that is high if any tour position is occupied by more than one city.
        - The **C term** is a penalty that is high if the total number of visited stops is not exactly *N*.
        - The **D term** is the actual objective. It adds up the distances (dxy) for all adjacent cities in the proposed tour.
    - By evolving to minimize this total energy *E*, the network tries to satisfy all the constraints (A, B, C) while also finding a tour that makes the distance term (D) as small as possible.

---

10. Digital Simulation of a Neurodynamical Model

Let's select the **Hodgkin-Huxley model**, a landmark achievement in computational neuroscience that describes in great detail how action potentials (nerve impulses) are generated.

Digital Simulation Tools: A powerful and common choice is the **Python** programming language combined with its scientific libraries: **NumPy** for numerical operations, **SciPy** for advanced scientific computing (especially its differential equation solvers), and **Matplotlib** for plotting the results.

The model consists of four coupled differential equations. A simulation would proceed as follows:

1. **Studying Dynamics (The "What")**:
    - **Implementation**: The four equations are coded into a Python function that calculates the rate of change for each variable (voltage *V*, and gating variables *m, h, n*) given their current values.
    - **Integration**: Using a numerical solver like `scipy.integrate.odeint`, we can simulate the neuron's behavior over time. We provide initial conditions and a stimulus (e.g., a brief pulse of input current) and the solver computes the state of the neuron at each time step.
    - **Visualization**: We can then use Matplotlib to plot the membrane potential *V* versus time. This plot would visually demonstrate the model's dynamics, showing us *what* the neuron does: it might remain at a resting potential, fire a single sharp action potential in response to the stimulus, or fire a train of them.
2. **Studying Stability (The "Why")**:
    - **Analysis**: To understand *why* the neuron has a stable resting potential, we perform stability analysis. We would first find the equilibrium points of the system by setting all derivatives to zero. Then, at the resting potential equilibrium, we would compute the Jacobian matrix and its eigenvalues.
    - **Insight**: Finding that all eigenvalues have negative real parts would mathematically prove that the resting state is stable, explaining why the neuron returns to rest after small perturbations.
3. **Studying Attractor Properties (The "Global Behavior")**:
    - **Phase Portraits**: To understand the global behavior, we can create phase portraits. For instance, we could simulate the model and plot the membrane potential *V* on the x-axis and one of the gating variables (like *m*) on the y-axis. The trajectory would trace a path in this space.
    - **Insight**: For a single action potential, this trajectory would show a large loop, starting from the resting state (a fixed point attractor), swinging out, and then spiraling back in. If the neuron is firing continuously, the trajectory would settle into a stable closed loop—a **limit cycle attractor**, visually representing the oscillatory nature of the firing. This analysis reveals the overall structure of the system's possible behaviors.

---

11. Practical Applications of Neurodynamical Systems

Neurodynamical systems, particularly attractor networks like Hopfield's model, have inspired several powerful real-world applications.

1. **Associative (Content-Addressable) Memory**:
    - **Application**: This is the most direct application, used in tasks like restoring corrupted images, recognizing patterns, and building robust memory systems.

    - **How it Works**: A Hopfield network stores patterns (e.g., digital images of faces) as attractors. The key feature is that memory is **content-addressable**. You don't access a memory by an address or index; you access it with a piece of its content. If you provide a noisy or incomplete picture of a face as input, the network's dynamics will automatically pull the state towards the closest complete face pattern stored in its memory, effectively cleaning up the noise and completing the image.
2. **Combinatorial Optimization**:
    - **Application**: Solving extremely difficult logistical and design problems, such as the Traveling Salesman Problem (TSP), optimal scheduling, and designing the layout of components on a computer chip (VLSI design).
    - **How it Works**: These problems are often "NP-hard," meaning finding the absolute best solution is computationally infeasible for large instances. A Hopfield network provides a way to find a very good, though not always perfect, solution very quickly. The problem's cost function is mapped to the network's energy function. The network's rapid, parallel relaxation into a low-energy state provides a high-quality approximate solution much faster than many traditional algorithms could.
3. **Medical Image Analysis and Classification**:
    - **Application**: Assisting radiologists in identifying anomalies in medical scans, such as classifying tumors in MRI or CT images.
    - **How it Works**: More advanced attractor network models can be trained on large datasets of labeled medical images. The network learns to create distinct attractors corresponding to different diagnostic categories (e.g., "healthy," "benign tumor," "malignant tumor"). When a new scan is fed to the network, the system's state evolves towards one of these pre-learned attractors. The attractor it settles into provides a proposed classification, acting as a "second opinion" to aid the human expert.

---

12. Analysis of the Dynamical System dx/dt=−x+x3

We are given the differential equation describing a simple dynamical system:

dtdx=−x+x3
a) Find the equilibrium points.

Equilibrium points are the states where the system does not change, meaning dtdx=0.

−x+x3=0
We can factor out an *x*:

x(x2−1)=0
Further factoring the difference of squares gives:

x(x−1)(x+1)=0
This equation is true if any of the factors are zero. Thus, the equilibrium points are x = 0, x = 1, and x = -1.

b) Analyze the stability of each equilibrium point.

We use the linearization method. We define the function f(x)=−x+x3 and find its derivative, which tells us how a small perturbation will grow or shrink.

f′(x)=dxd(−x+x3)=−1+3x2
Now, we evaluate the sign of this derivative at each equilibrium point:

- For the point x = 0:
  f′(0)=−1+3(0)2=−1.
  Since the derivative is negative, any small perturbation from x=0 will shrink, pulling the system back to the equilibrium. Therefore, the equilibrium point at x = 0 is stable.
- For the point x = 1:
  f′(1)=−1+3(1)2=−1+3=2.
  Since the derivative is positive, any small perturbation will be amplified, pushing the system away from the equilibrium. Therefore, the equilibrium point at x = 1 is unstable.
- For the point x = -1:
  f′(−1)=−1+3(−1)2=−1+3=2.
  Since the derivative is positive, this point is also unstable. A small nudge will cause the system to move away from x=-1. Therefore, the equilibrium point at x = -1 is unstable.

---

13. Asynchronous Update of a Hopfield Network

We are given the weight matrix W and the initial state Sinitial:

W= 01−1101−110 ,Sinitial= 1−11
The update rule is Si(t+1)=sgn(hi), where the input hi=∑jWijSj. The sgn function returns +1 for positive input, -1 for negative input, and leaves the state unchanged for zero input. We will update the neurons one by one in the order 1, 2, 3.

Initial State: S=[1,−1,1]

Cycle 1

- **Update Neuron 1**:
  - Input h1=W12S2+W13S3=(1)(−1)+(−1)(1)=−1−1=−2.
  - New state for neuron 1: S1new=sgn(−2)=−1.
  - The network state is now S=[−1,−1,1].

- **Update Neuron 2** (using the newly updated state):
    - Input $h2 = W21S1 + W23S3 = (1)(-1) + (1)(1) = -1 + 1 = 0$.
    - New state for neuron 2: $S2new = sgn(0) = S2old = -1$ (no change).
    - The network state remains $S = [-1, -1, 1]$.
- **Update Neuron 3** (using the current state):
    - Input $h3 = W31S1 + W32S2 = (-1)(-1) + (1)(-1) = 1 - 1 = 0$.
    - New state for neuron 3: $S3new = sgn(0) = S3old = 1$ (no change).
    - The network state remains $S = [-1, -1, 1]$.

Cycle 2

The state at the beginning of cycle 2 is $[-1, -1, 1]$. Let's check if any further updates will occur.

- **Update Neuron 1**: $h1 = (1)(-1) + (-1)(1) = -2 \Rightarrow S1 = sgn(-2) = -1$. (No change).
- **Update Neuron 2**: $h2 = (1)(-1) + (1)(1) = 0 \Rightarrow S2 = -1$. (No change).
- **Update Neuron 3**: $h3 = (-1)(-1) + (1)(-1) = 0 \Rightarrow S3 = 1$. (No change).

Since a full cycle of updates results in no changes, the network has reached a stable state.

The attractor state the network settles into is **[-1, -1, 1]**.

---

14. Verifying a Pattern is an Attractor

We are asked to verify that the pattern $p = [1, -1, 1, 1]$ is a stable attractor for a Hopfield network trained on it using the Hebbian rule, $Wij = pipj$ for $i \neq j$, and $Wii = 0$.

First, let's establish the pattern components: $p1 = 1, p2 = -1, p3 = 1, p4 = 1$.

A pattern is a stable attractor if, for every neuron *i* in the pattern, its state pi is equal to the sign of the input it receives from all other neurons. That is, we must verify $pi = sgn(\sum_{j \neq i} Wijpj)$ for all *i*.

Let's check each neuron:

- **Check Neuron 1 (p1=1)**:
    - Input $h1 = W12p2 + W13p3 + W14p4$
    - $h1 = (p1p2)p2 + (p1p3)p3 + (p1p4)p4 = (1 \cdot -1)(-1) + (1 \cdot 1)(1) + (1 \cdot 1)(1) = 1 + 1 + 1 = 3$.
    - $sgn(3) = 1$, which is equal to p1. **(Stable)**.
- **Check Neuron 2 (p2=−1)**:
    - Input $h2 = W21p1 + W23p3 + W24p4$
    - $h2 = (p2p1)p1 + (p2p3)p3 + (p2p4)p4 = (-1 \cdot 1)(1) + (-1 \cdot 1)(1) + (-1 \cdot 1)(1) = -1 - 1 - 1 = -3$.
    - $sgn(-3) = -1$, which is equal to p2. **(Stable)**.
- **Check Neuron 3 (p3=1)**:
    - Input $h3 = W31p1 + W32p2 + W34p4$
    - $h3 = (p3p1)p1 + (p3p2)p2 + (p3p4)p4 = (1 \cdot 1)(1) + (1 \cdot -1)(-1) + (1 \cdot 1)(1) = 1 + 1 + 1 = 3$.

    - sgn(3)=1, which is equal to p3. **(Stable)**.
- **Check Neuron 4 (p4=1)**:
    - Input h4=W41p1+W42p2+W43p3
    - h4=(p4p1)p1+(p4p2)p2+(p4p3)p3=(1⋅1)(1)+(1⋅−1)(−1)+(1⋅1)(1)=1+1+1=3.
    - sgn(3)=1, which is equal to p4. **(Stable)**.

Since every neuron in the pattern is stable, the entire pattern p=[1,−1,1,1] is a stable fixed point of the network's dynamics and is therefore an attractor.

---

15. Analysis of a Two-Neuron Hopfield Network

We are given the weight matrix for a two-neuron network:

W=(0330)
i. Find the eigenvalues of this weight matrix.

To find the eigenvalues (λ), we solve the characteristic equation det(W−λI)=0, where I is the identity matrix.

det(0−λ330−λ)=0
det(−λ33−λ)=0
The determinant is (−λ)(−λ)−(3)(3)=λ2−9.

λ2−9=0
λ2=9
Solving for λ, we get the two eigenvalues: λ1=3 and λ2=−3.

ii. Using the eigenvalues, determine if the network is likely to have stable equilibrium points.

While the eigenvalues are an interesting property of the matrix, the stability of a Hopfield network's equilibria is more directly determined by the fundamental properties of the weight matrix itself.

For a Hopfield network with asynchronous updates, convergence to stable fixed points (attractors) is **guaranteed** if the weight matrix meets two conditions:

1. **Symmetry**: Wij=Wji. Our matrix has W12=3 and W21=3, so it is symmetric.
2. **Zero Diagonal**: Wii=0. Our matrix has W11=0 and W22=0, so this condition is met.

Because this network's weight matrix satisfies both crucial conditions, it is **guaranteed to have stable equilibrium points**. The symmetry and zero-diagonal structure ensure that there is a well-defined energy function that the network's dynamics will always descend, forcing it to settle in a stable minimum.

16. Stability Analysis of dx/dt=−3x+x2

The given neural system is described by the equation:

dtdx=−3x+x2
i. Find the equilibrium points.

We find the equilibrium points by setting the rate of change to zero:

−3x+x2=0
Factoring out an *x*, we get:

x(x−3)=0
This gives two equilibrium points: x = 0 and x = 3.

ii. Use a Lyapunov function candidate V(x)=x2 to analyze stability.

A Lyapunov function helps us determine stability by acting like an energy function. We need to check its derivative with respect to time, dtdV.

Using the chain rule:

$$\frac{dV}{dt} = \frac{dV}{dx} \frac{dx}{dt}$$We have $\frac{dV}{dx} = 2x$ and $\frac{dx}{dt} = -3x + x^2$.$$\frac{dV}{dt} = (2x)(-3x + x^2) = -6x^2 + 2x^3 = 2x^2(x-3)$$

- **Analysis at the equilibrium point x = 0**:
    1. **Is V(x) valid?**: At x=0, V(0)=02=0. For any other x=0, V(x)=x2>0. So, V(x) is positive-definite around x=0. It is a valid candidate.
    2. **What is the sign of dtdV?**: We analyze dtdV=2x2(x−3) in the neighborhood of x=0. For any small non-zero *x*, the 2x2 term is positive. The (x−3) term will be negative (e.g., if x=0.1, it's -2.9).
    3. Therefore, dtdV is negative in the neighborhood of x=0.
    4. **Conclusion**: Since we found a valid Lyapunov function whose derivative is negative, the equilibrium point at **x = 0 is asymptotically stable**.
- **Analysis at the equilibrium point x = 3**:
    1. **Is V(x) valid?**: At x=3, V(3)=32=9. A Lyapunov function candidate *must* be zero at the equilibrium point in question. Since V(3)=0, V(x)=x2 is **not a valid Lyapunov function** for testing the stability of the point x=3.
    2. **Alternative (Informal) Analysis**: Although it's not a formal Lyapunov proof, we can still examine the sign of dtdV=2x2(x−3) near x=3. For a value just above 3 (e.g., x=3.1), the term (x−3) is positive, making dtdV positive. This indicates that the "energy" increases as the system moves away from 3, which is characteristic of an unstable point. The point **x = 3 is unstable**.

16 (bis). Analysis of Network Activity

The activity of a two-neuron network is given by the explicit time-dependent equations:

$x(t)=2-2e-0.5t$
$y(t)=3-3e-0.5t$
The term $e-0.5t$ represents a transient effect that decays over time.

i. Calculate the values of x(t) and y(t) at t=2 and t=4.

- **At t = 2**:
  - $e-0.5(2)=e-1\approx0.36788$
  - $x(2)=2-2(e-1)\approx2-2(0.36788)=2-0.73576=1.26424$
  - $y(2)=3-3(e-1)\approx3-3(0.36788)=3-1.10364=1.89636$
- **At t = 4**:
  - $e-0.5(4)=e-2\approx0.13534$
  - $x(4)=2-2(e-2)\approx2-2(0.13534)=2-0.27068=1.72932$
  - $y(4)=3-3(e-2)\approx3-3(0.13534)=3-0.40602=2.59398$

ii. Determine if the system is converging to a point attractor and identify its coordinates.

To determine the long-term behavior, we need to find the limit of x(t) and y(t) as time t approaches infinity.

As $t\to\infty$, the exponential term $e-0.5t$ approaches 0, because the exponent becomes a very large negative number.

- $\lim_{t\to\infty}x(t)=\lim_{t\to\infty}(2-2e-0.5t)=2-2(0)=2$
- $\lim_{t\to\infty}y(t)=\lim_{t\to\infty}(3-3e-0.5t)=3-3(0)=3$

Yes, the system's state approaches a single, fixed point as time goes on. It is therefore converging to a **point attractor**.

The coordinates of this attractor are **(2, 3)**.

---

17. Analysis of a Two-Neuron Recurrent Network

The network's dynamics are given by the system of linear differential equations:

$dtdx=-x+ay$
$dtdy=-y+bx$
i. Find the equilibrium points for this network (with a=1, b=2).

At equilibrium, all rates of change are zero. We set dtdx=0 and dtdy=0, substituting the given parameters a=1 and b=2.

1. $-x+(1)y=0 \Longrightarrow -x+y=0$
2. $-y+(2)x=0 \Longrightarrow -y+2x=0$

This is a system of two linear equations. From equation (1), we can express y in terms of x:

$y=x$
Now, we substitute this result into equation (2):

$-(x)+2x=0$
$x=0$
Since we know $y=x$, if $x=0$, then y must also be 0.

The only solution is $(x,y)=(0,0)$. Therefore, the network has a single equilibrium point at the origin, (0, 0).

ii. Determine the effect on the equilibrium points if a is increased to 3.

We now repeat the analysis with the new parameter $a=3$ (while $b=2$ remains the same).

The system of equations becomes:

1. $-x+(3)y=0 \Longrightarrow -x+3y=0$
2. $-y+2x=0$

From equation (1), we express x in terms of y:

$x=3y$
Now, substitute this into equation (2):

$$-y + 2(3y) = 0$$$$-y + 6y = 0$$$$5y = 0 \implies y = 0$$
Since $x=3y$, if $y=0$, then $x=3(0)=0$.

The only solution is again $(x,y)=(0,0)$.

Effect: Increasing the parameter *a* from 1 to 3 has **no effect on the location** of the equilibrium point. It remains at **(0, 0)**. While this change does not move the equilibrium point, it significantly alters the dynamics *around* that point, changing its stability properties.