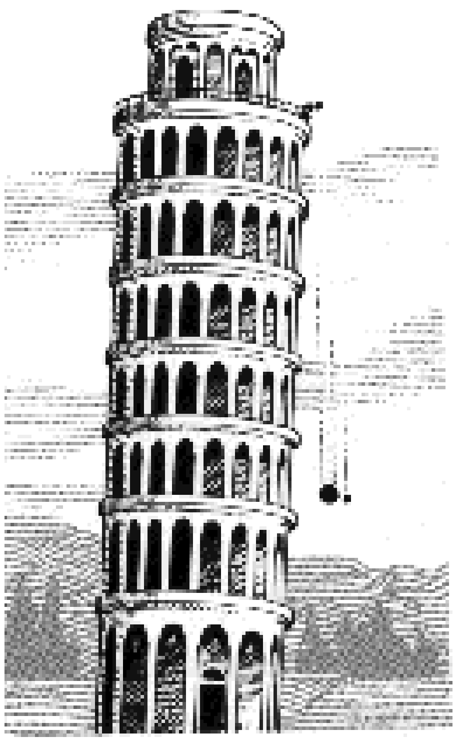
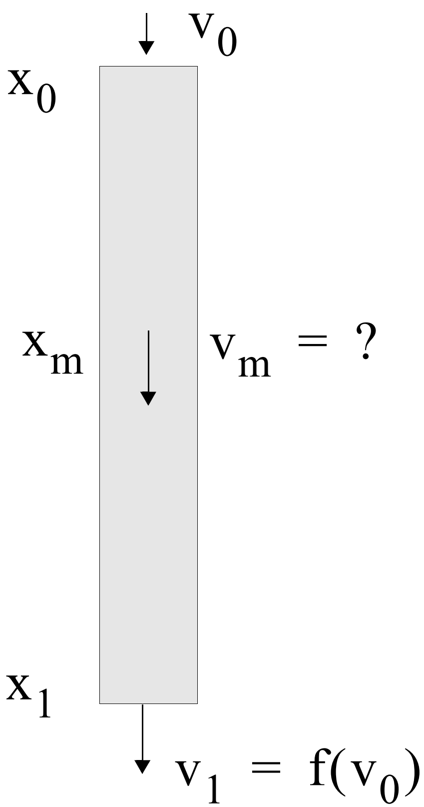
# Functional Equations & Neural Networks for Time Series Interpolation

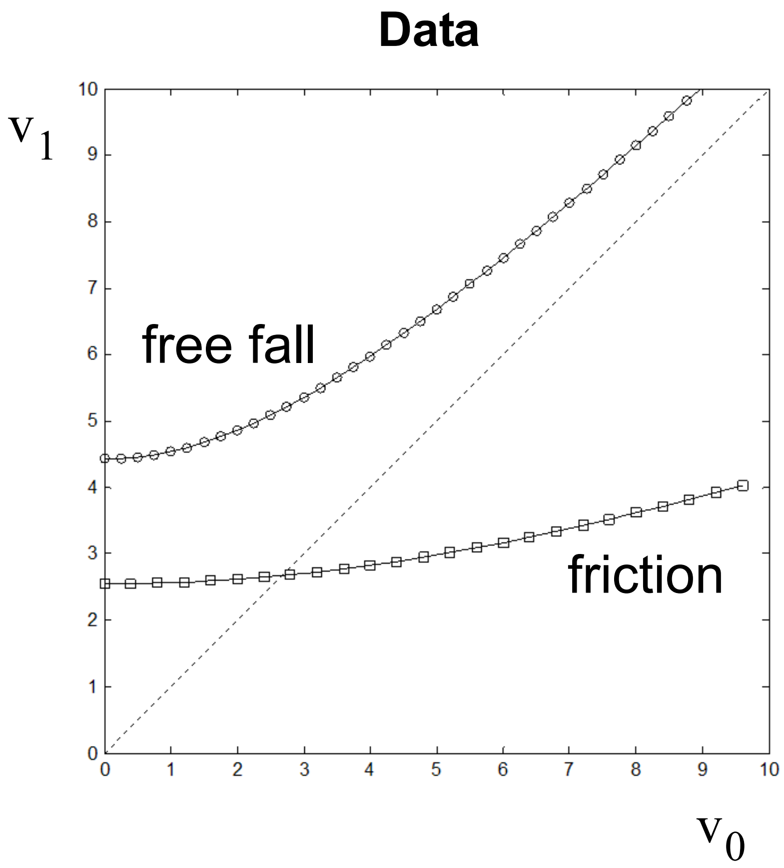
Lars Kindermann, AWI
Achim Lewandowski, OEFAI

# An old experiment

Drop an object with different speeds $v_0$ and measure the speed at the ground $v_1$






Question: What's the speed $v_m$ after half the way at $x_m$?

# Solving with traditional Physics

**Free Fall:**

$$\text{Theory: } \frac{\partial^2 x}{\partial t^2} = g \quad \Rightarrow$$

$$\text{Model: } v_1 = f(v_0) = \sqrt{v_0^2 + 2g\Delta x} \text{ with data fitted } g$$

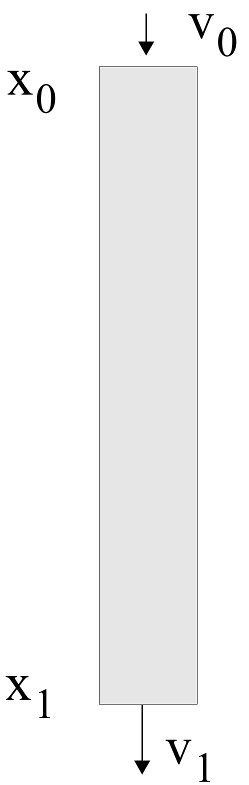
**With additional Friction:**

$$\text{Theory: } \frac{\partial^2 x}{\partial t^2} = g - k_1\left|\frac{\partial x}{\partial t}\right| - k_2\left|\frac{\partial x}{\partial t}\right|^2 - f\left(\frac{\partial x}{\partial t}\right) \quad \Rightarrow$$

Model: Integrate numerically and fit $g$ and $k$ - already a non-trivial Problem!

# A Data-based Aproach

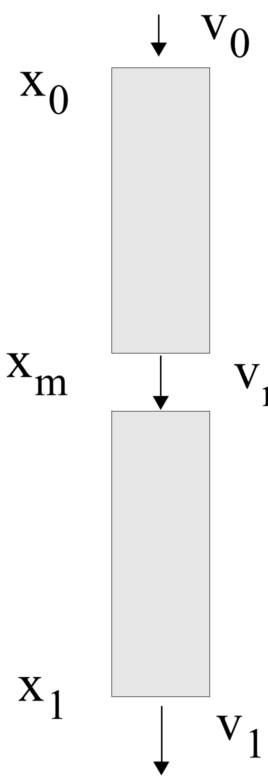
**Theory: Assume translation invariance**




$$v_1 = f(v_0)$$

$\Rightarrow$

divide into
two equal
steps...




$$v_m = \varphi(v_m)$$

$$v_1 = \varphi(v_m) = \varphi(\varphi(v_($$

$$\boxed{\varphi(\varphi(v)) = f(v)}$$

and solve this functional equation for $\varphi$

# A Functional Equation

$$\varphi(\varphi(x)) = f(x)$$

A solution $\varphi$ of this equation is a kind of *square root* of the function $f$.

- If $f(x): \mathbb{R}^n \to \mathbb{R}^n$ is a *function*, we look for another function $\varphi(x)$ which composed with itself equals $f$: $\varphi(\varphi(x)) = f(x)$

Because the self-composition of a function $f(f(x)) = f^2(x)$ is also called "iteration", the square root of a function is usually called its ***iterative root.***

$$\varphi^n(x) = f^m(x)$$

is solved by the ***fractional iterates*** of a function $f$:

$$\varphi(x) = f^{m/n}(x)$$

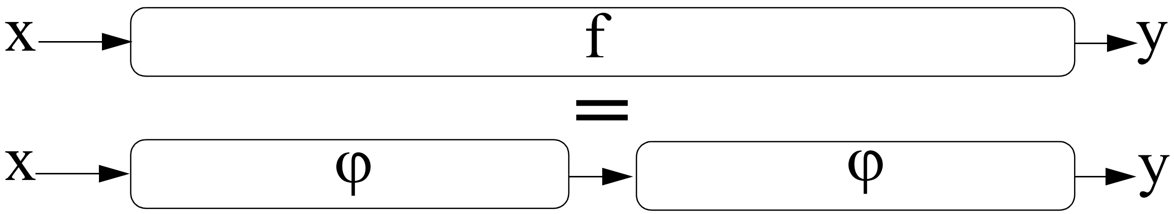
# A Functional Equation

$$\varphi(\varphi(x)) = f(x)$$

A solution $\varphi$ of this equation is called a *square root* of $f$.

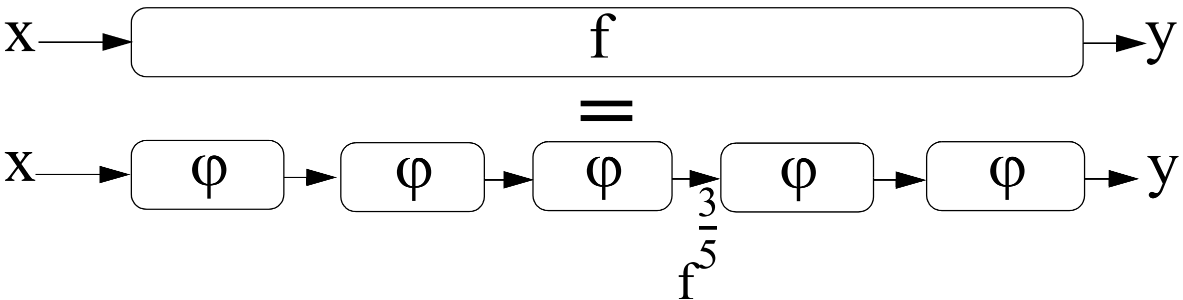
- If $f(x): \mathbb{R}^n \to \mathbb{R}^n$ is a *function*, we look for another function $\varphi(x)$ which composed with itself equals $f$: $\varphi(\varphi(x)) = f(x)$

Because the self-composition of a function $f(f(x)) = f^2(x)$ is also called “iteration”, the square root of a function is usually called its ***iterative root.***




$\varphi^n(x) = f^m(x)$ is solved by the ***fractional iterates*** of a function $f$:

$$\varphi(x) = f^{m/n}(x)$$



# Generalized Iteration

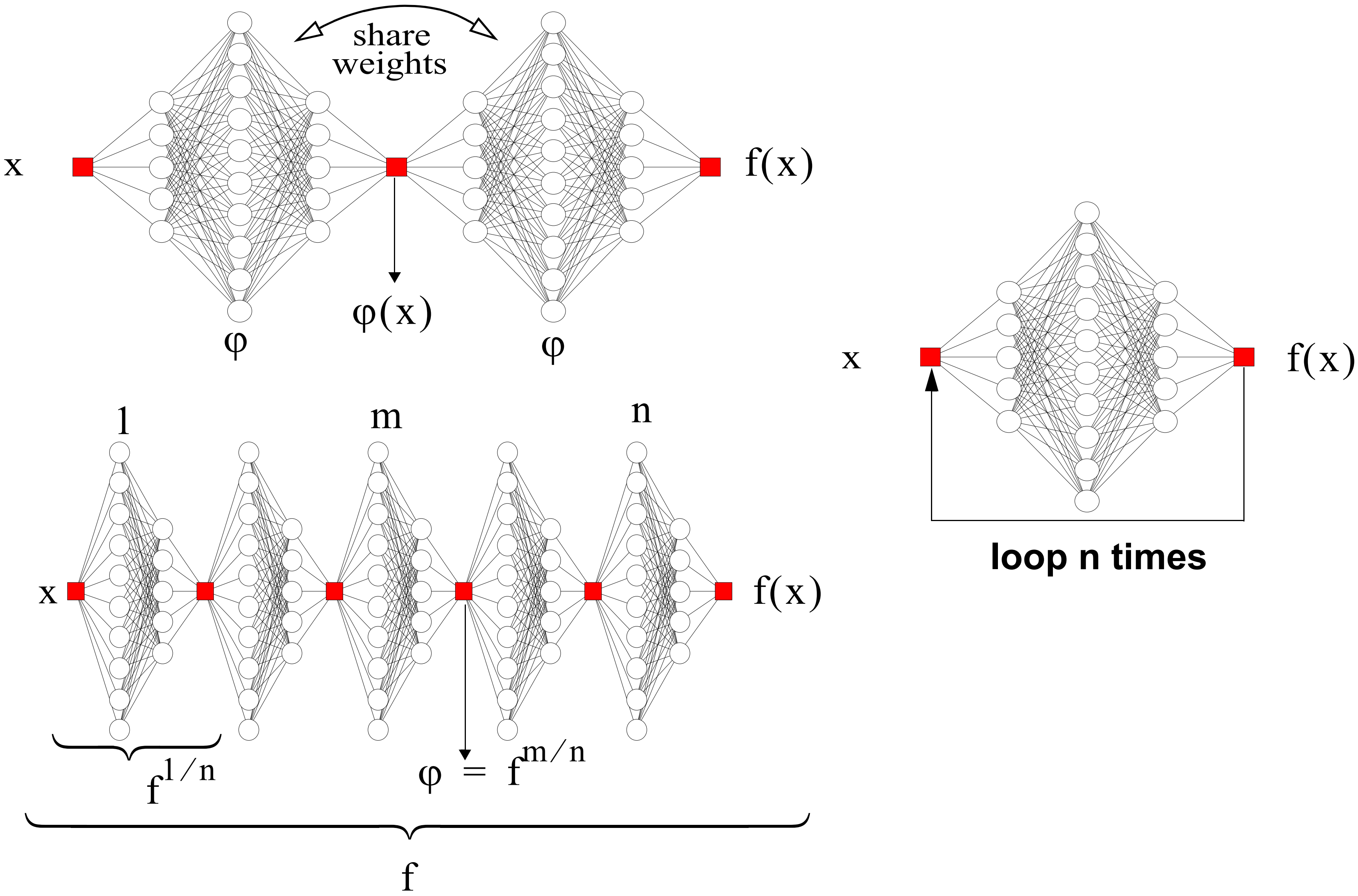
The exponential notation of the iteration of functions $f^n(x)$ can be extended beyond integer exponents:

- $f^1$ means $f$
- $f^n$ for positive integers $n$ are the well known iterations of $f$
- $f^0$ denotes the identity function, $f^0(x) = x$
- $f^{-1}$ is the inverse funktion of $f$
- $f^{-n}$ is the $n$-th iteration of the inverse of $f$
- $f^{1/n}$ is the $n$-th iterative root of $f$
- $f^{m/n}$ is the $m$-th iteration of the $n$-th iterative root or *fractional iterate* of $f$

The family $f^t(x)$ forms the *continuous iteration group* of $f$.

Within this the *translation equation* $f^{a+b}(x) = f^a(f^b(x))$ is satisfied.

# Map this to a Network

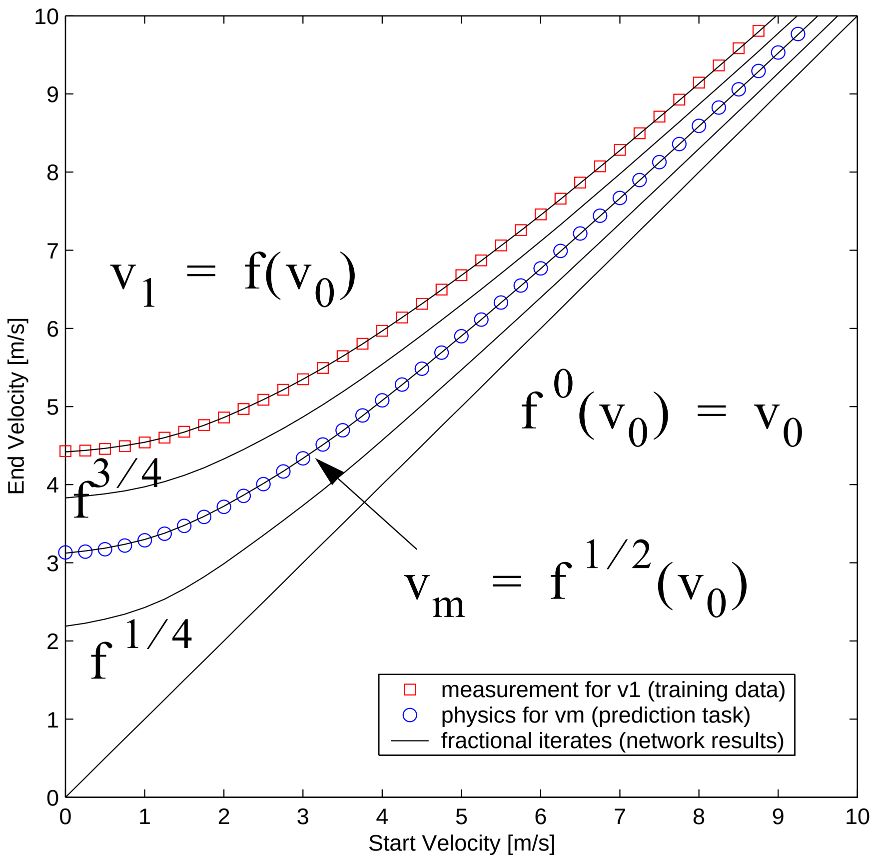


# Training Methods

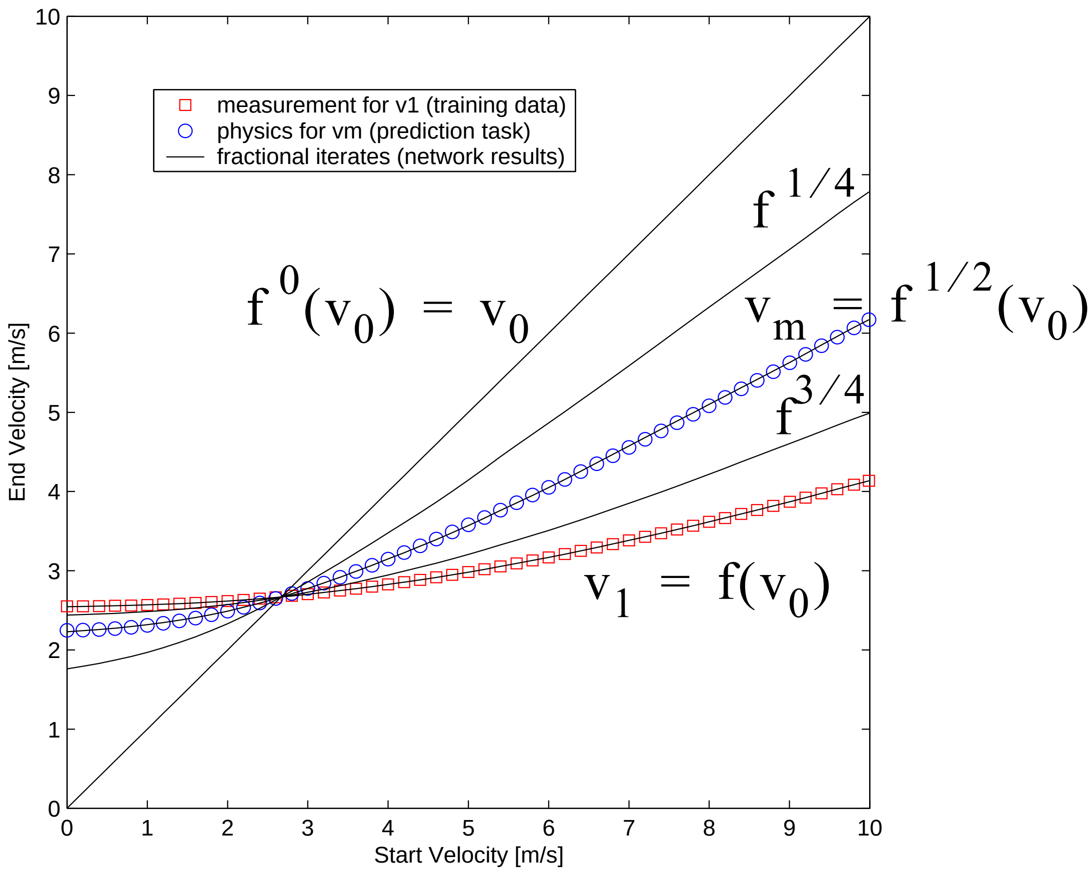
- **Weight Copy:** Train only the last layer and copy the weights continously backwards
- **Weight Sharing:** Initialize corresponding weights with equal values and sum up all $\delta w_i$ delivered by the network learning rule
- **Weight Coupling:** Start with different values and let the corresponding weights of the iteration layers approach each other by a term like $\delta w_i = \alpha(w_j - w_i)$
- **Regularization:** Add a penalty term to the error function which assigns an error to the weight-differences to regularize the network. This allows to utilize second order gradient methods like quasi Newton for faster training.
- **Exact Gradient:** Compute the exact gradients for an iterated Network

# Results for „The Fall“

no friction

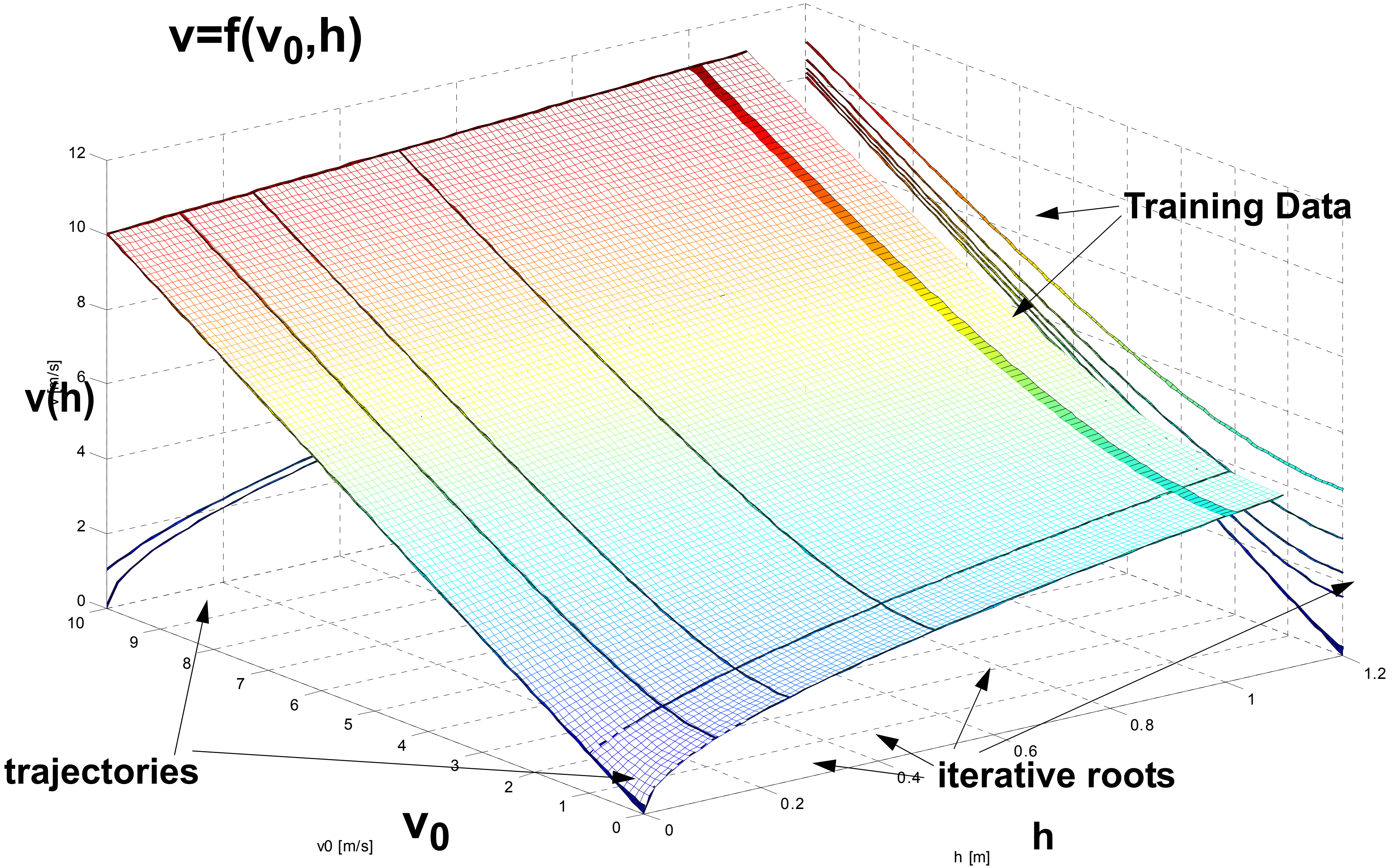



with friction




The Network results are conform to the laws of physics up to a mean error of $10^{-6}$

# The Embedding Problem

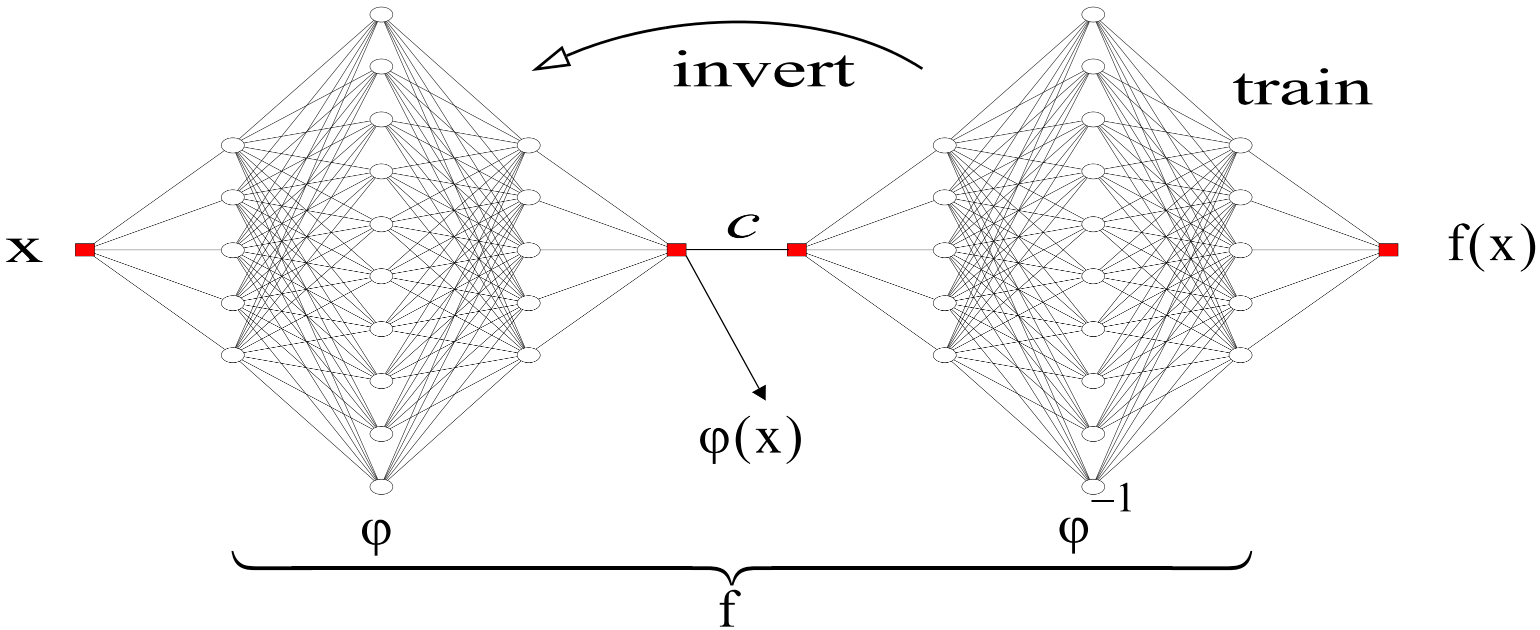


# The Schröder Equation

$$\varphi(f(x)) = c\varphi(x)$$

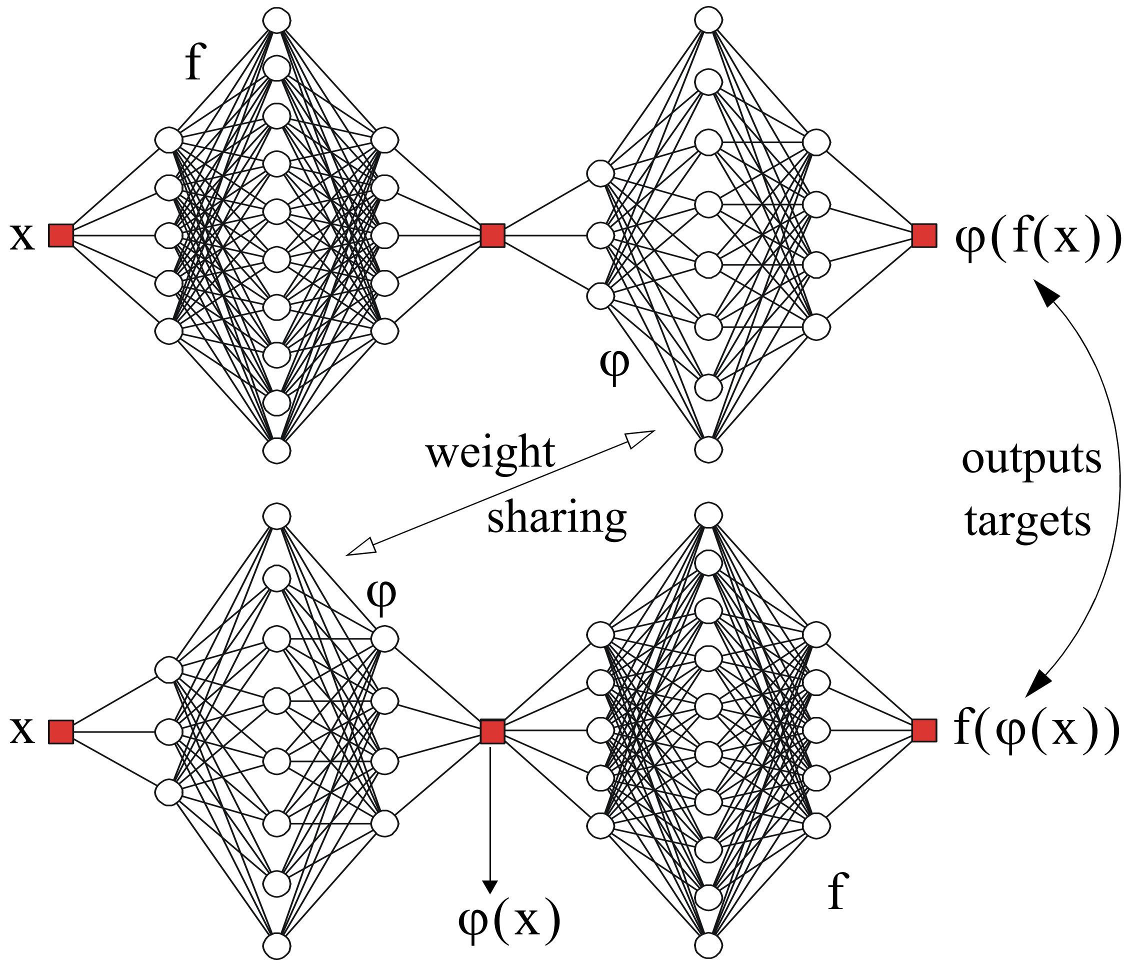
One of the most important functional equations:
The Eigenvalue problem of functional calculus.

Transform to: $f(x) = \varphi^{-1}(c\varphi(x))$



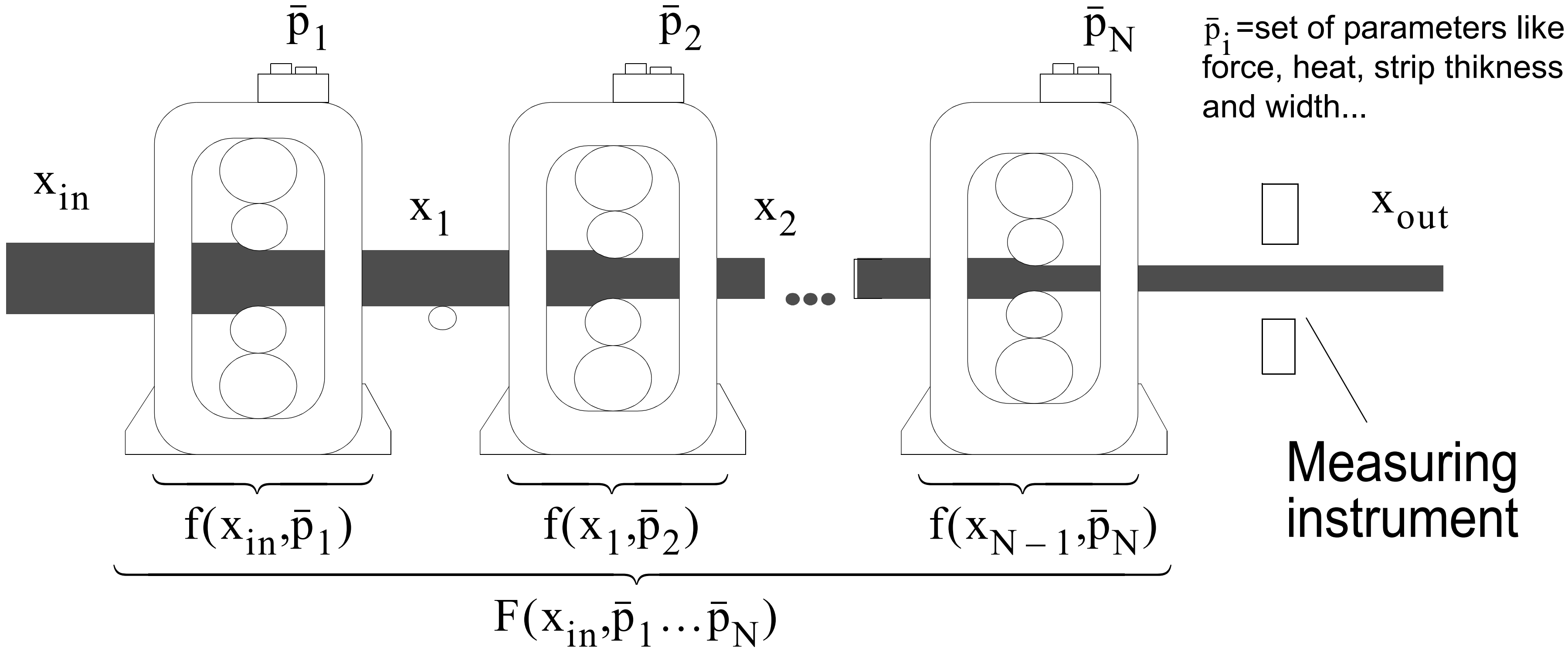
# Commuting Functions

$$\varphi(f(x)) = f(\varphi(x))$$



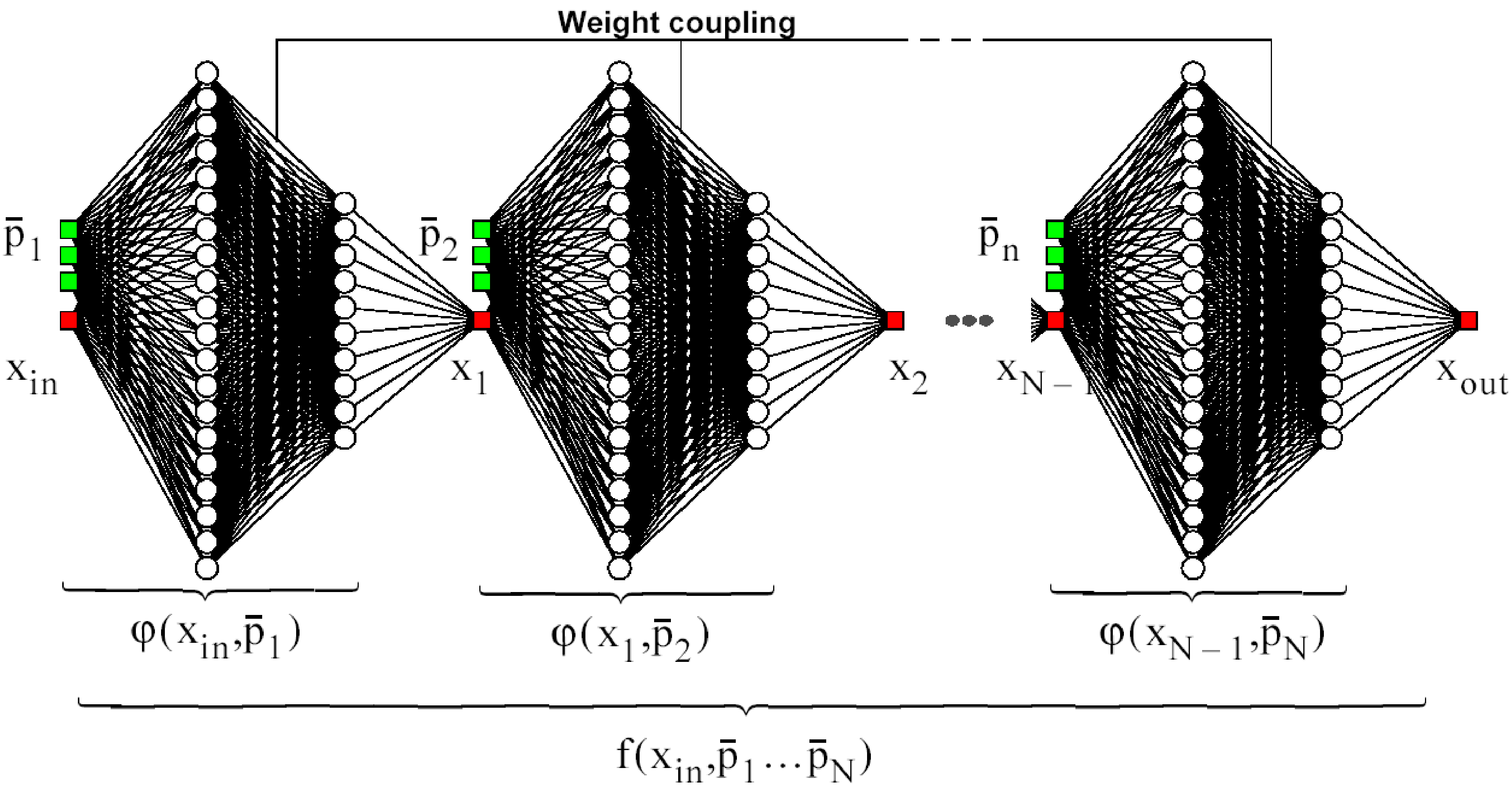

MESTA

# Steel Mill Model




- The steel bands are processed by *N* identical stands in a row
- $x_{in}$, $p_i$ are known and $x_{out}$ can be measured
- $x_{out} = F(x_{in}, \bar{p}_1 \ldots \bar{p}_N) = f(\ldots f(f(x_{in}, \bar{p}_1), \bar{p}_2) \ldots, \bar{p}_N)$

# Steel Mill Network



# Timeseries Interpolation

For a given autoregressive Box-Jenkins AR(n) timeseries $x_t = \sum_{k=1}^{n} a_k x_{t-k}$, we

define the function $F: R^n \to R^n$ which maps the vector of the last n samples

$\vec{x}_{t-1} = [x_{t-1}, \ldots, x_{t-n}]$ one step into the future $\vec{x}_t = [x_t, x_{t-1}, \ldots, x_{t-(n-1)}]$ as

$$F = \begin{bmatrix} a_1 & a_2 & \ldots & a_n \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and can simply write $\vec{x}_t = F \cdot \vec{x}_{t-1}$ now.

The discrete time evolution of the the system can be calculated using the matrix powers of F: $\vec{x}_{t+n} = F^n \cdot \vec{x}_{t-1}$.

# Using Generalized Matrix Powers

This autoregressive system is called linear embeddable if the matrix power $F^t$ exists also for all real $t \in R^+$. This is the case if $F$ can be decomposed into $F = S \cdot A \cdot S^{-1}$ with $A$ being a diagonal matrix consisting of the eigenvalues $\lambda_i$ of $F$ and $S$ being an invertible square matrix which columns are the eigenvectors of $F$. Additionally all $\lambda_i$ must be non-negative to have a linear and *real* embedding, otherwise we will get a *complex* embedding.

Then we can obtain $F^t = S \cdot A^t \cdot S^{-1}$ with $A^t = \begin{bmatrix} \lambda_1{}^t & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & \lambda_n{}^t \end{bmatrix}$

Now we have a continuous function $\vec{x}(t) = F^t \cdot \vec{x}_0$ and the interpolation of the original time series $x(t)$ consists of the first element of $\vec{x}$.

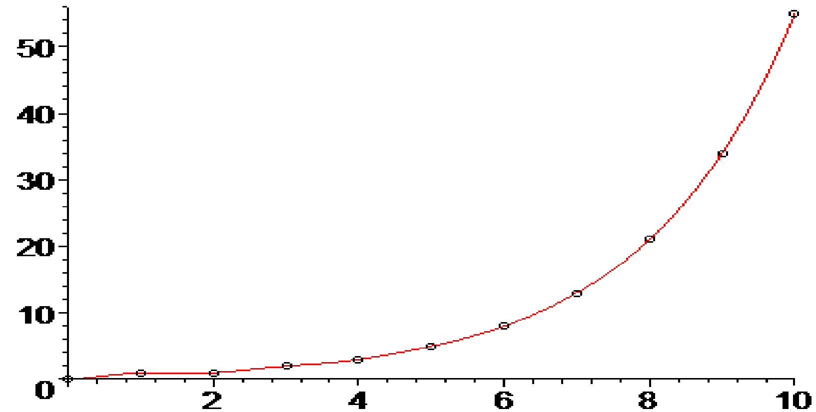
# Example: A continuous Fibonacci Function

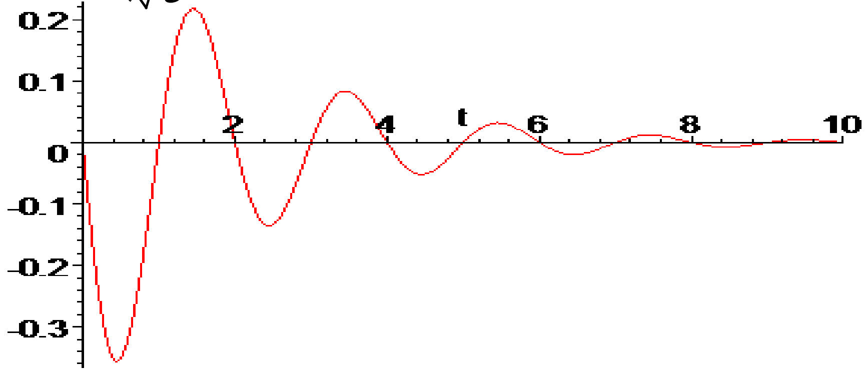
The Fibonacci series $x_0 = 0,\ x_1 = 1,\ x_t = x_{t-1} + x_{t-2}$ is generated by

$F = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ and $\vec{x}_1 = [1, 0]$. By eigenvalue decomposition of $F$ we get

$$\vec{x}_{t+1} = F^t \vec{x}_1 = S A^t S^{-1} \vec{x}_1 = \begin{bmatrix} \frac{1+\sqrt{5}}{2} & \frac{1-\sqrt{5}}{2} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \left(\frac{1+\sqrt{5}}{2}\right)^t & 0 \\ 0 & \left(\frac{1-\sqrt{5}}{2}\right)^t \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{5}} & \frac{1}{2} - \frac{1}{2\sqrt{5}} \\ -\frac{1}{\sqrt{5}} & \frac{1}{2} + \frac{1}{2\sqrt{5}} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

That is Binet's formula in the first component $x_t = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^t - \left(\frac{1-\sqrt{5}}{2}\right)^t\right]$

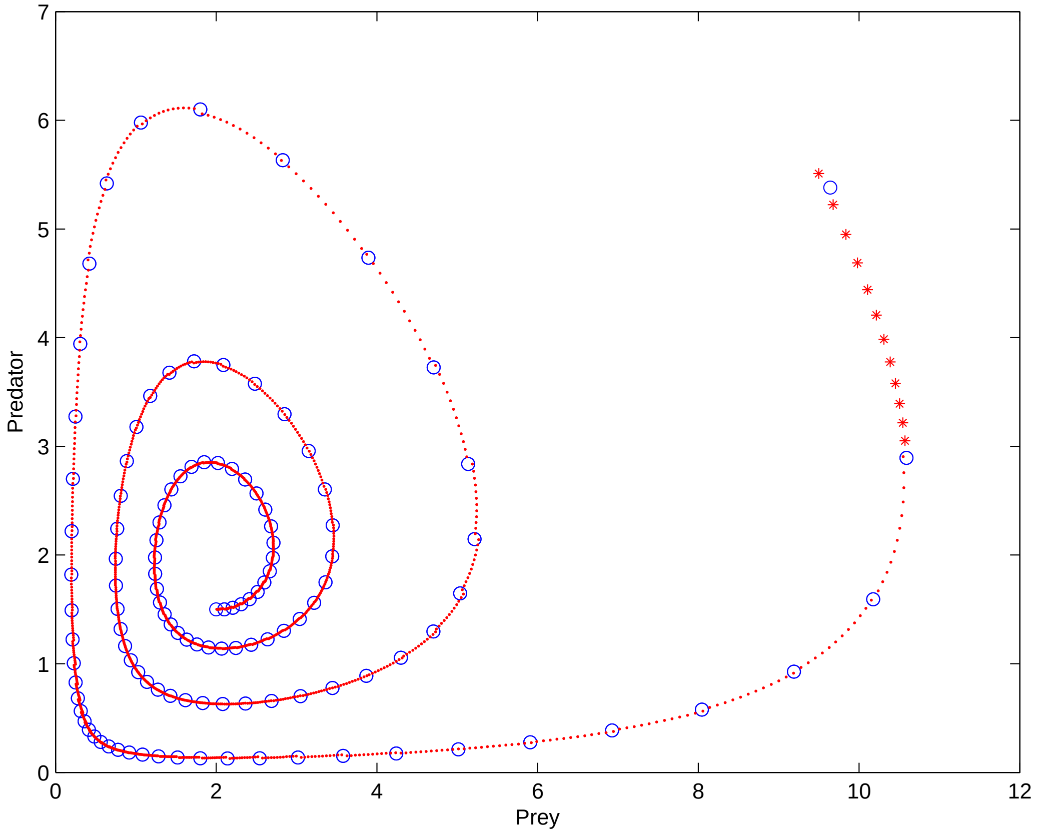


# Nonlinear Example

A time series of ***yearly*** snapshots from a discrete non linear Lotka-Volterra type predator - prey system ($x$ = hare, $y$ = lynx) is used as training data:

$x_{t+1} = (1 + a - b\, y_t)\, x_t$ and $y_{t+1} = (1 - c + d\, x_t)\, y_t$




From these samples we calculate the ***monthly*** population by use of a neural network based method to compute iterative roots and fractional iterates.

The given method provides a natural way to estimate not only the values over a year, but also to extrapolate arbitrarily smooth into the future.