Machine learning in astronomical time series

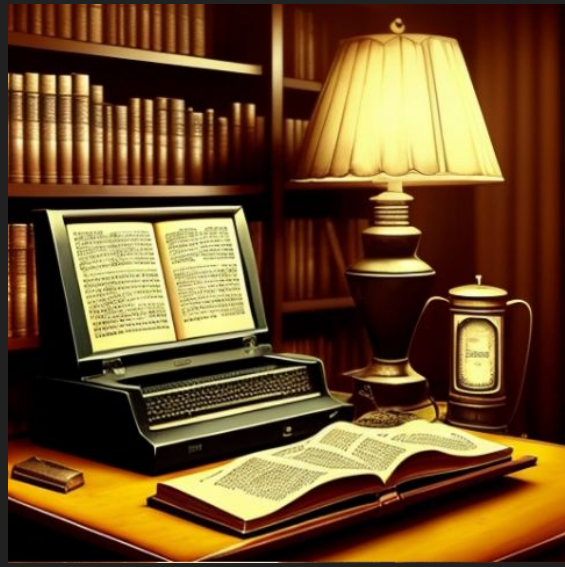
Niall Miller
University Of Hertfordshire
n.miller4@herts.ac.uk

What is Machine Learning (ML)?

Use maths to imitate the concept of learning by showing which values under a given frame work can produce a desired result.

- Least squares regression
- MCMC
- Gaussian Processes
- RNN
- GPT
- DDPM
- Scipy.optimize.curve_fit
- Automatically fitting “y = mx + c”

Prompt: A old style computer learning via reading a book



How do we teach a computer anything?

We have a value “x”

We have a function F(x)

We don't know what features for F(x) provide a given “x”

We give a computer a random guess for F(x)

We inform the computer how wrong it was (wrong = x - F(x))

The computer attempts F(x) again given knowledge of how wrong it was

We inform the computer how wrong it was (wrong = x - F(x))

The computer attempts F(x) again given knowledge of how wrong it was

We inform the computer how wrong it was (wrong = x - F(x))

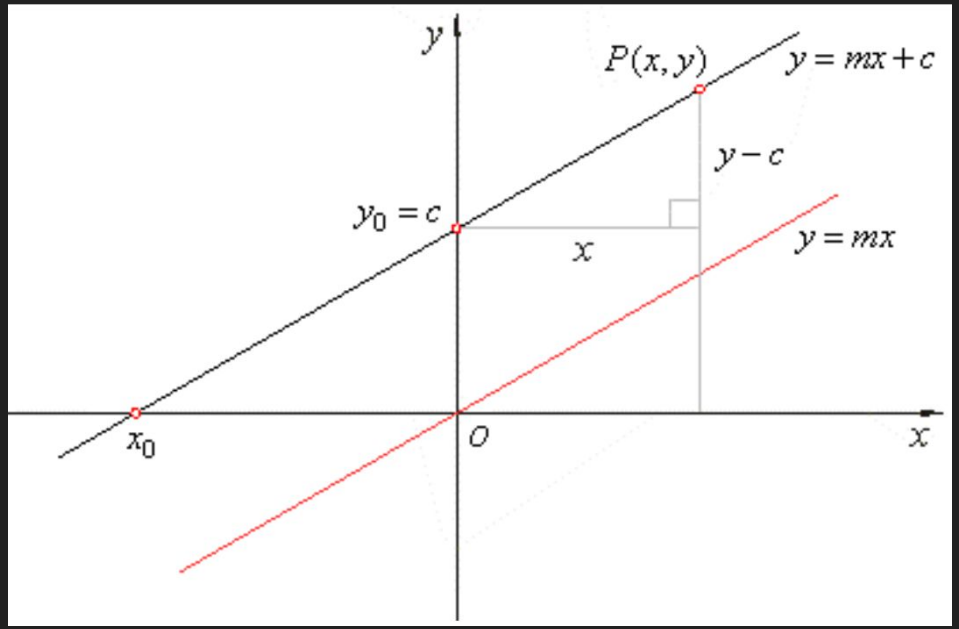
The computer attempts F(x) again given knowledge of how wrong it was

We inform the computer how wrong it was (wrong = x - F(x))

The computer attempts F(x) again given knowledge of how wrong it was

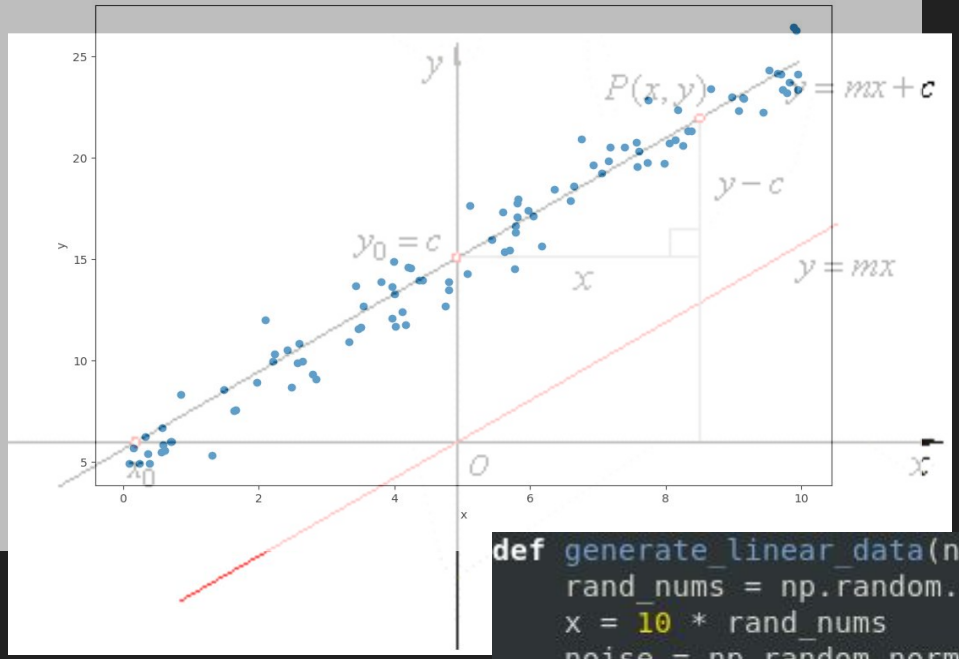
…We hope that eventually F(x) ≈ x

Linear Regression Example



Y = mX + c

Linear Regression Example

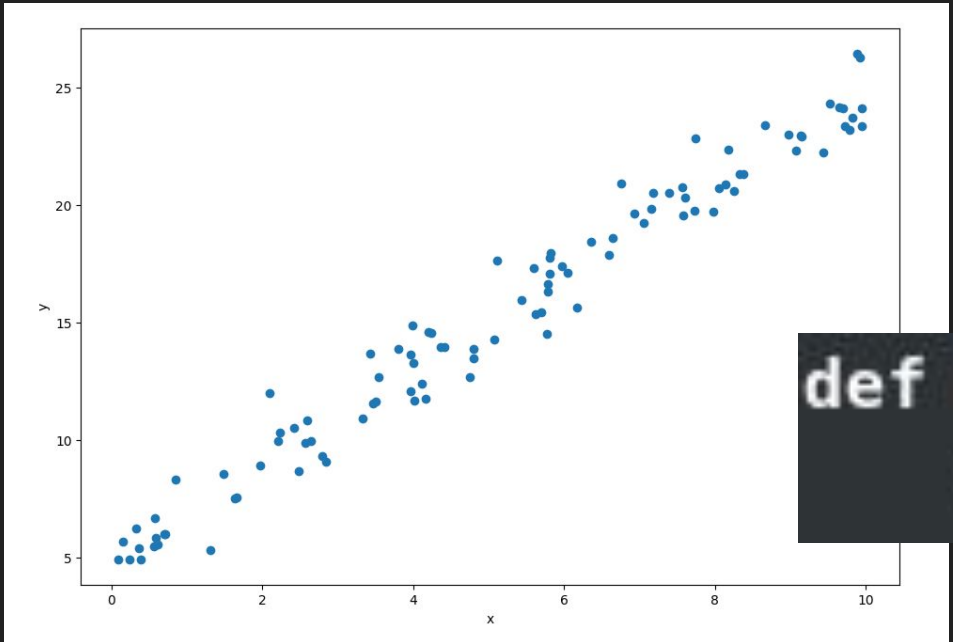


$$Y = mX + c$$

Or in code…

```
def generate_linear_data(num_points, gradient, intercept, noise_std_dev):
    rand_nums = np.random.rand(num_points)
    x = 10 * rand_nums
    noise = np.random.normal(0, noise_std_dev, num_points)
    y = linear_eq(x, gradient, intercept) + noise
    return x, y
```

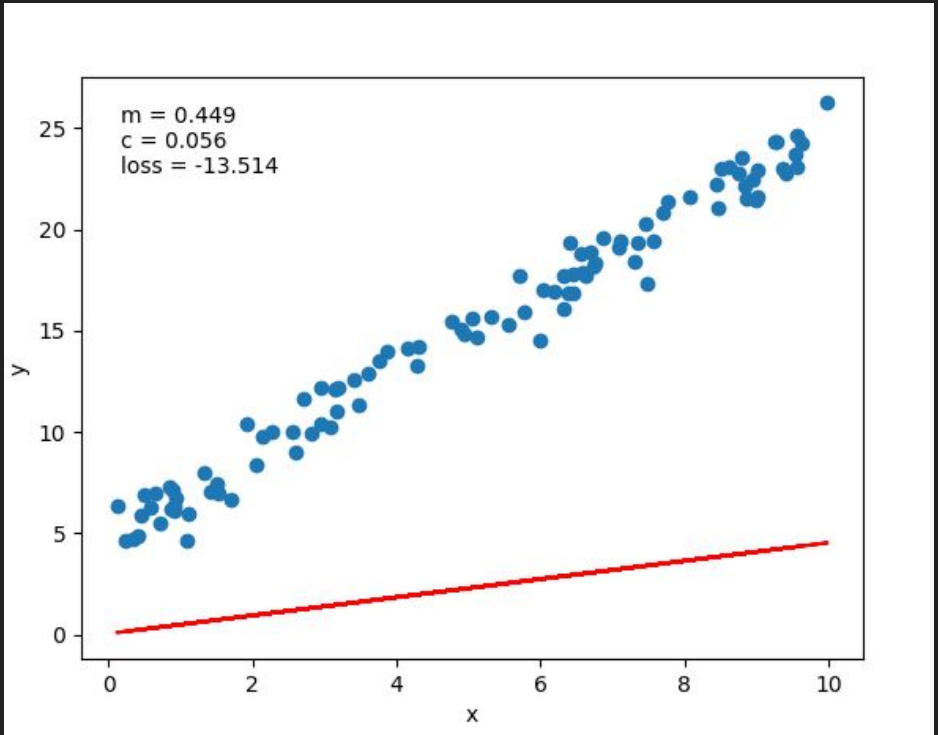
Linear Regression Example



We can try to get the computer to figure out what combination of ‘m’ and ‘c’ best represents this data

```
def linear_eq(X, m, c):
    return m*X + c
```

Linear Regression Example

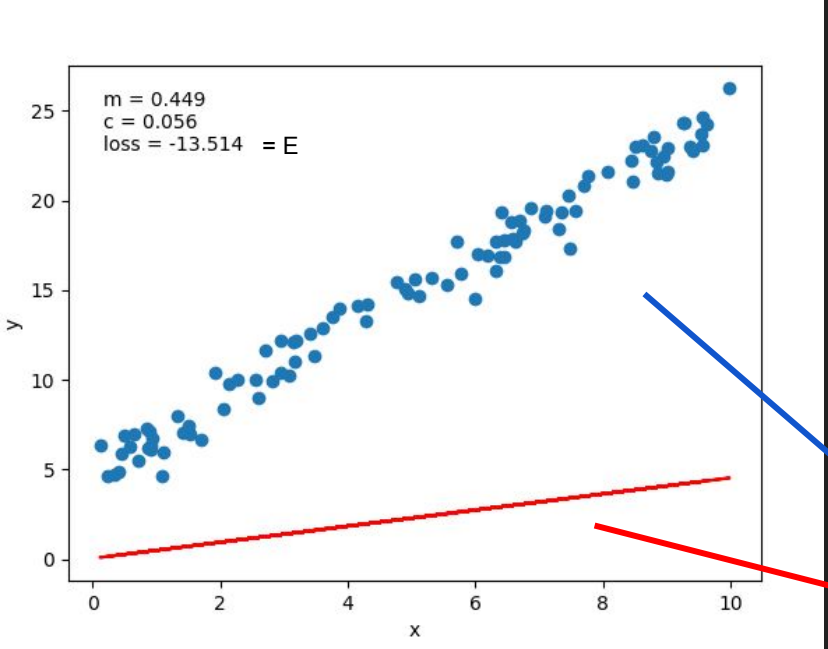


$Y_i = mx_i + c$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

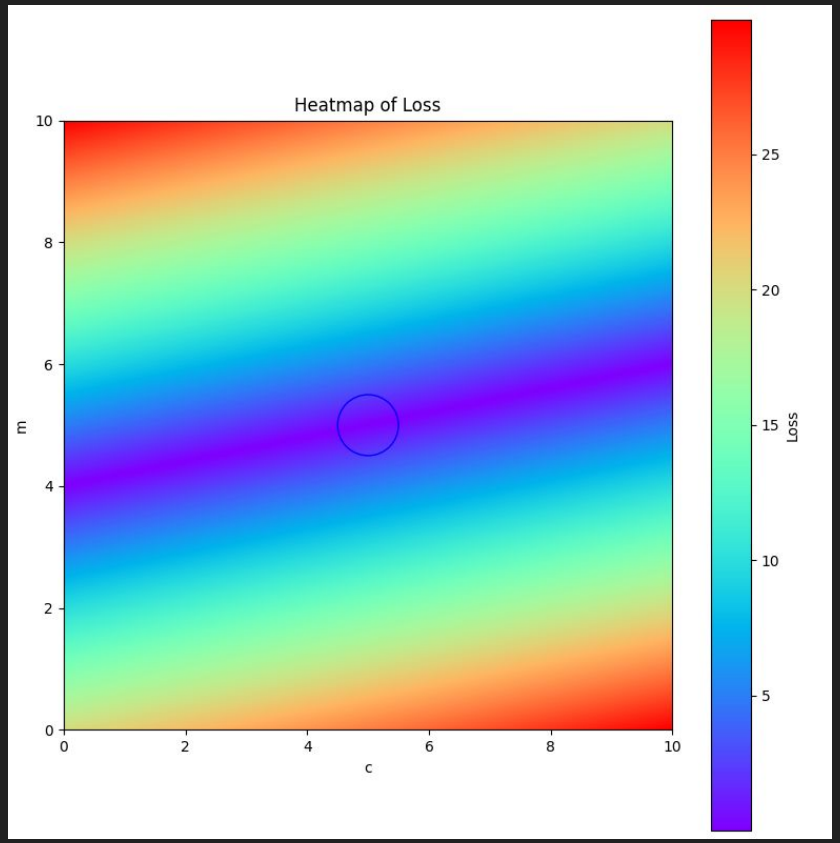
Linear Regression Example



$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

How do we inform the next step?



$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

dE/dm =

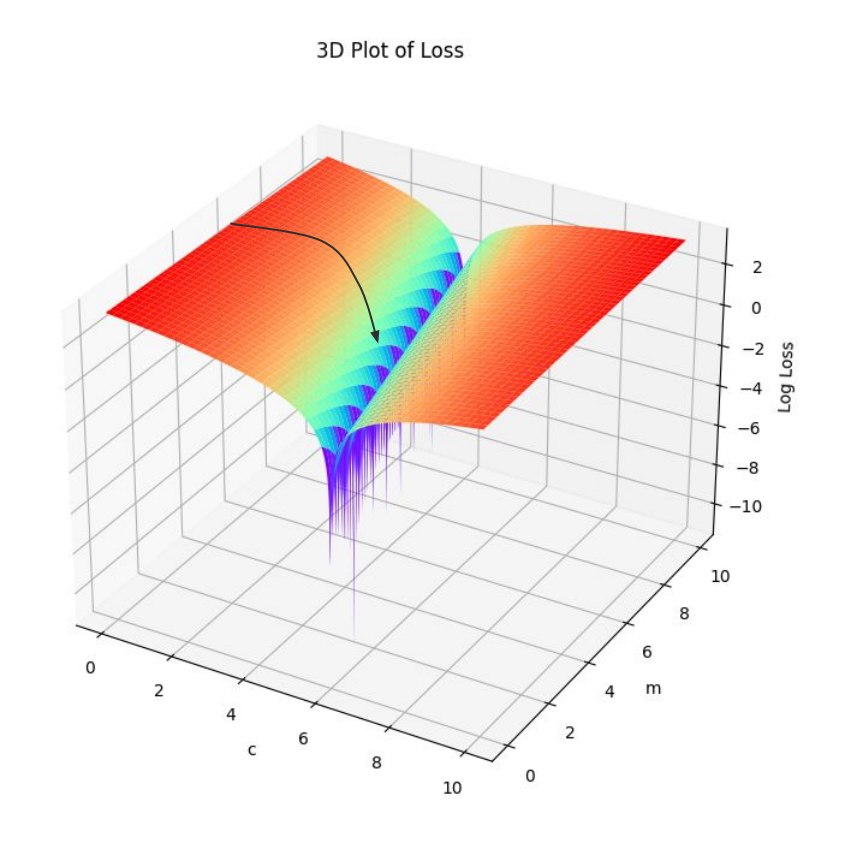
$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

dE/dc =

$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

How do we inform the next step?



dE/dm =

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

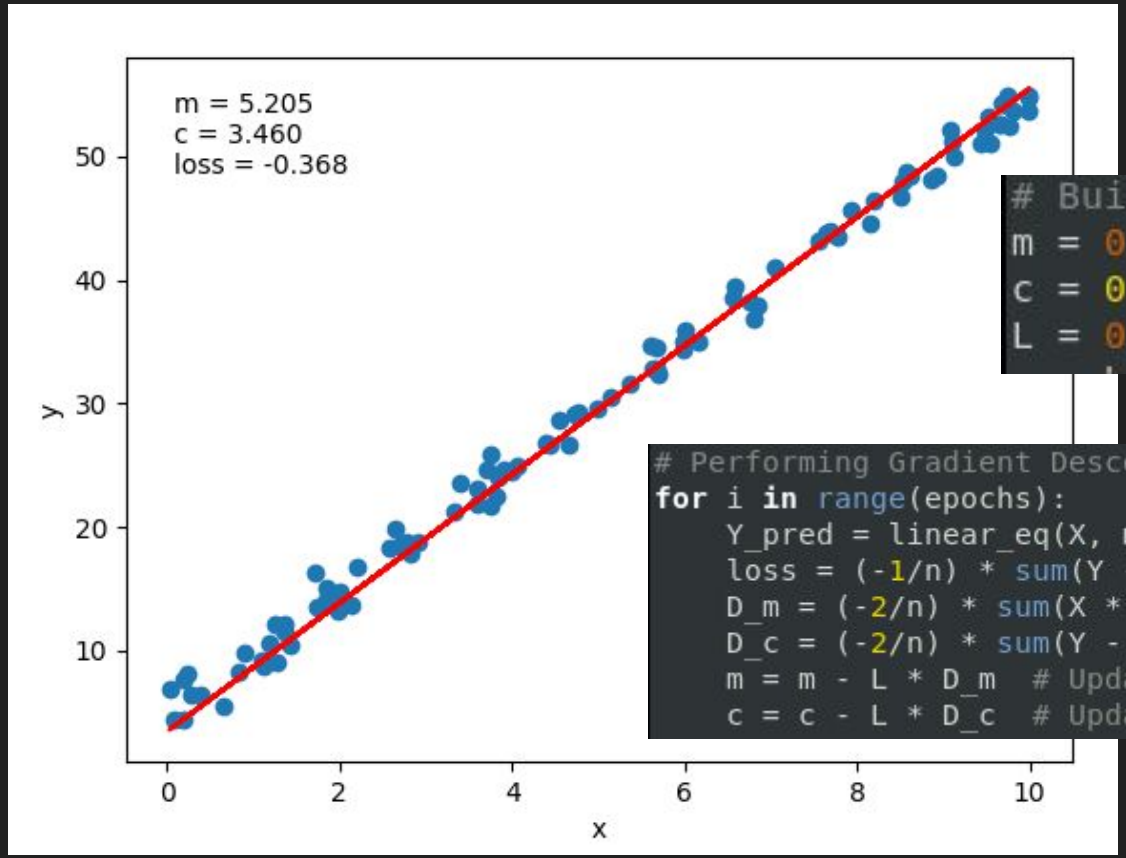
dE/dc =

$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)$$

$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

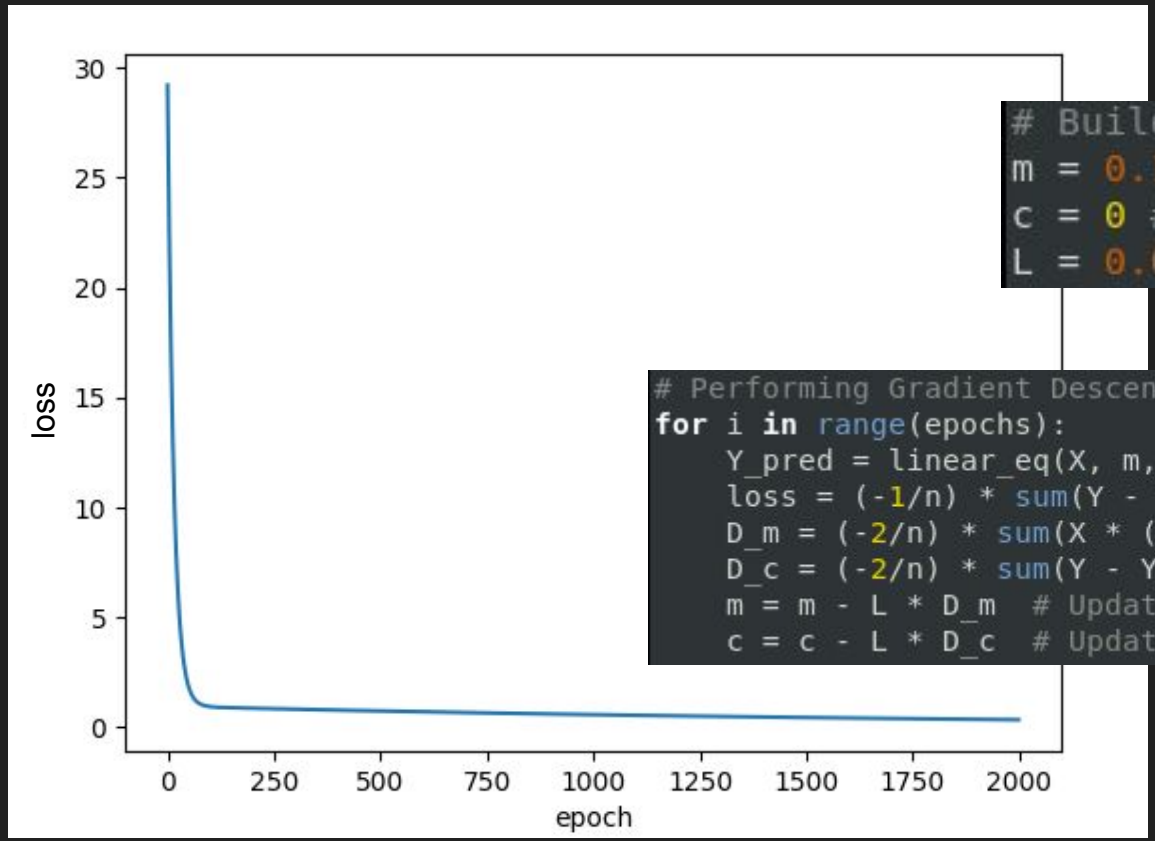
How do we inform the next step?




```
# Building the model
m = 0.1 #Initial guess for gradient
c = 0 #Initial guess for intercept
L = 0.001  # The learning Rate
```

```
# Performing Gradient Descent
for i in range(epochs):
    Y_pred = linear_eq(X, m, c)  # The current predicted value of Y
    loss = (-1/n) * sum(Y - Y_pred) #MSE loss
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c
```

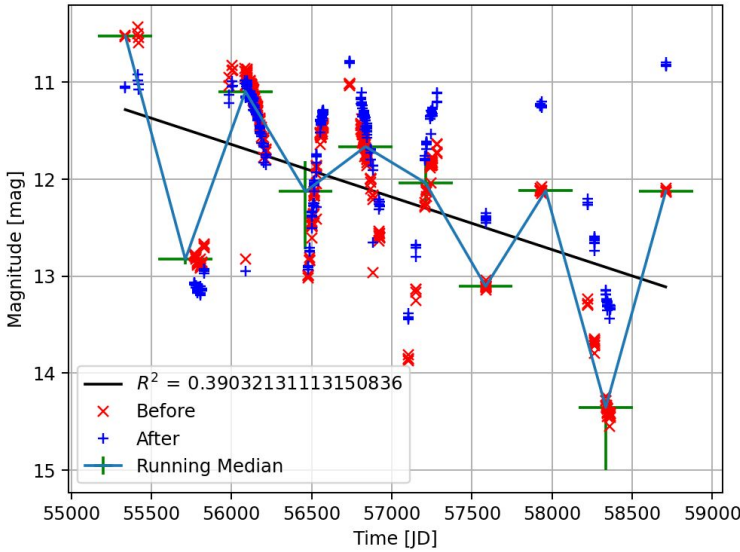
How do we inform the next step?

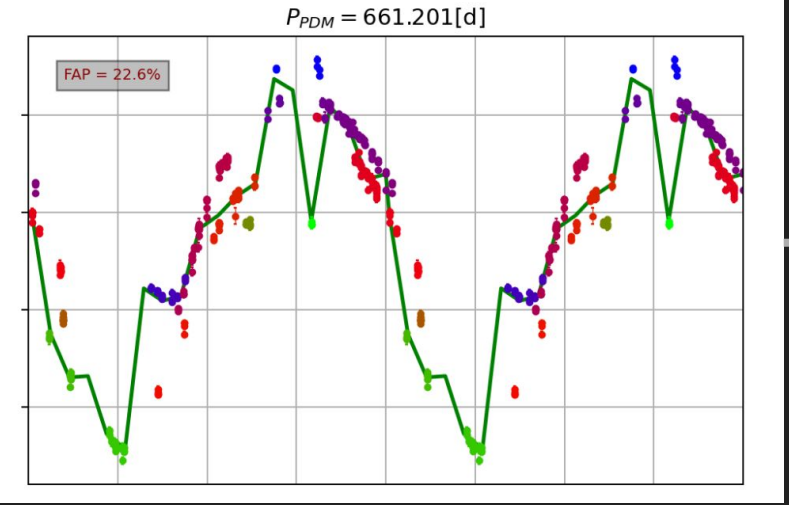



```
# Building the model
m = 0.1 #Initial guess for gradient
c = 0 #Initial guess for intercept
L = 0.001  # The learning Rate
```

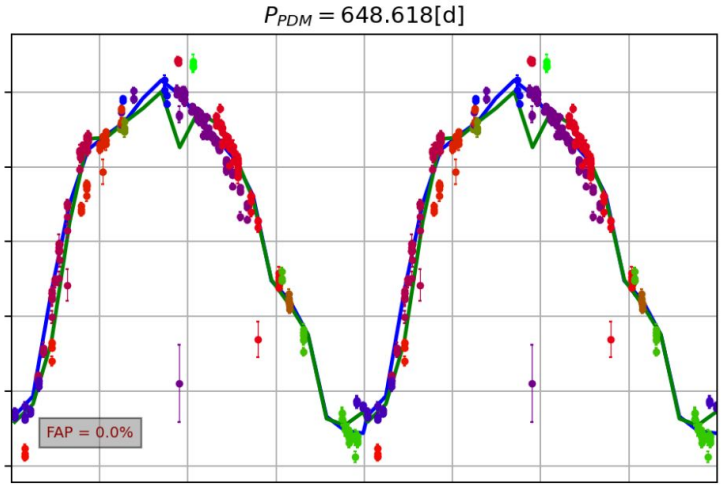
```
# Performing Gradient Descent
for i in range(epochs):
    Y_pred = linear_eq(X, m, c)  # The current predicted value of Y
    loss = (-1/n) * sum(Y - Y_pred) #MSE loss
    D_m = (-2/n) * sum(X * (Y - Y_pred))  # Derivative wrt m
    D_c = (-2/n) * sum(Y - Y_pred)  # Derivative wrt c
    m = m - L * D_m  # Update m
    c = c - L * D_c  # Update c
```

Do such simple tasks have any use?





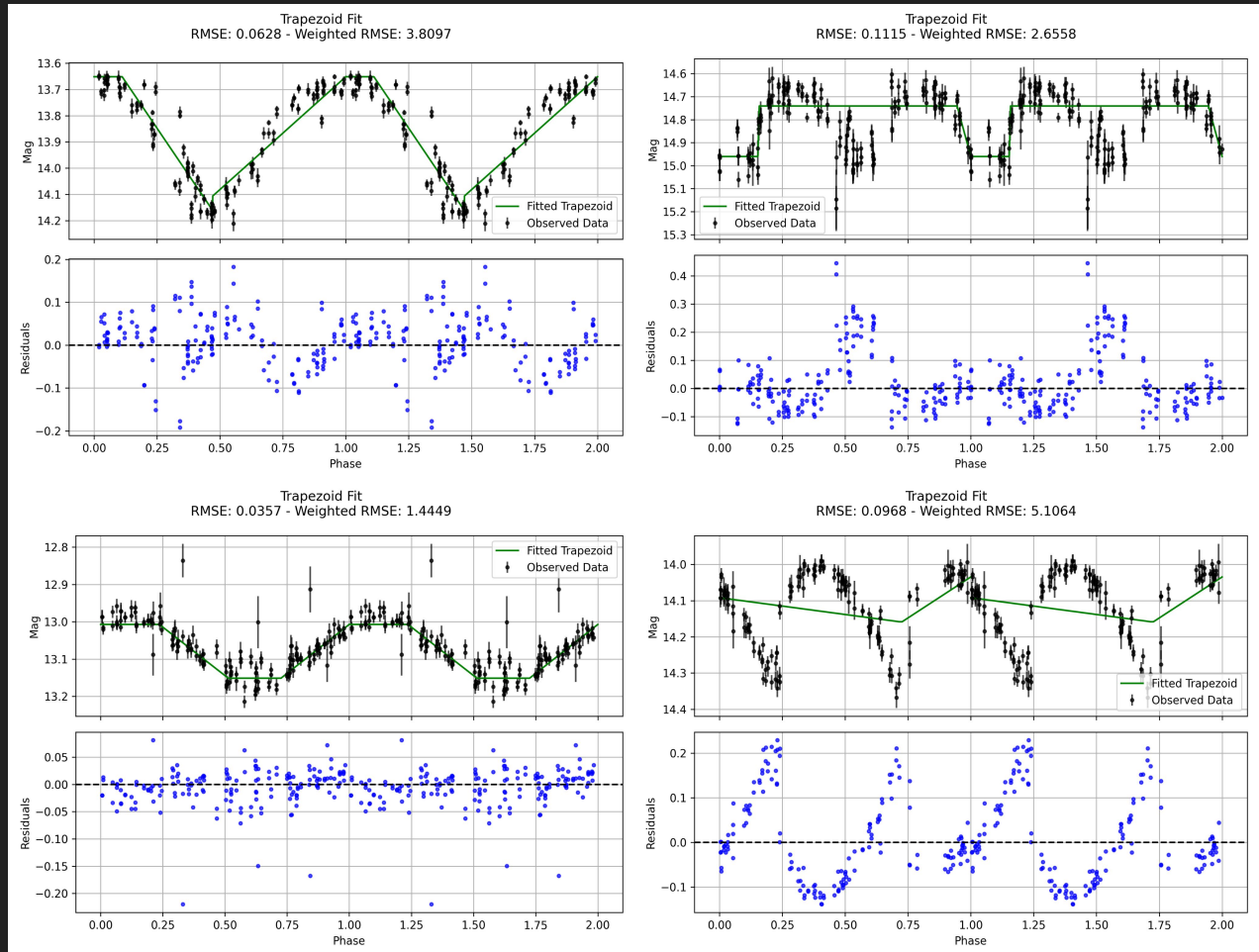
Remove linear trend



That was nice and easy, but that's mostly because this was a nice and easy problem…..

How about we use this fitting technique (with a trapezoid instead) to try to parameterise and classify stars?

It's pretty clear that this isn't sufficient and I, as a mere human, am unable to think of a reasonable equation/polynomial which would do this well. Nor do I know exactly how the noise behaves.



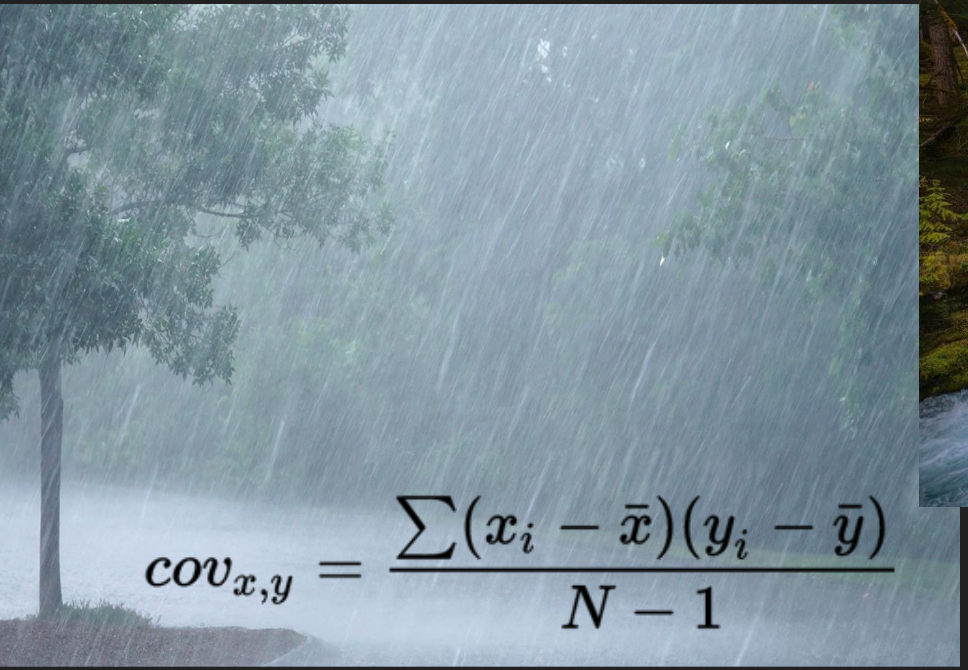
Gaussian processes are this but more….

Fundamentally different in 2 key ways

- GPs define a distribution over functions (kernels) rather than form a fixed set of parameters and thus provide us with posterior distributions (uncertainties)
- GPs fit to the *covariance* of the data and are thus much more flexible as a specific parametric form is not required.

In other words, GPs give us errors and are more flexible at capturing patterns.

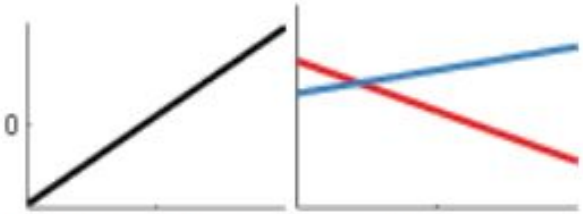
Covariance



$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

Linear Kernel

NB this kernel is literally just bayesian linear regression…



$$k_{\text{Lin}}(x, x') = \sigma_b^2 + \sigma_v^2(x - c)(x' - c)$$

Kernels

The Kernel Cookbook:

Advice on Covariance functions

by David Duvenaud

Update: I've turned this page into a chapter of my thesis.

If you've ever asked yourself: "How do I choose the covariance function for a Gaussian process?" this is the page for you. Here you'll find concrete advice on how to choose a covariance function for y

If you're looking for software to implement Gaussian process models, I recommend GPML for Matlab, or GPy for Python. These software packages deliberately do not provide a default kernel. You mi don't they include a sensible default?"

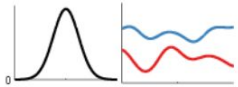
The answer is that the choice of kernel (a.k.a. covariance function) determines almost all the generalization properties of a GP model. You are the expert on your modeling problem - so you're the pe ough about kernels to choose a sensible one, read on.

Support Vector Machines

If your question is: "How do I choose a kernel for a Support Vector Machine"? Then a lot of the advice below might still be helpful, especially the first section on standard kernels. However, if you wan ve a hard time learning all the parameters by cross-validation. This is why most SVM kernels have only one or two parameters.

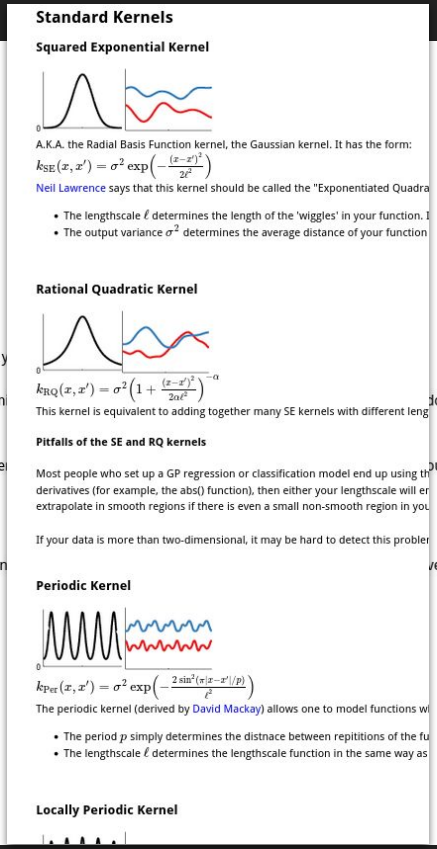
Standard Kernels

Squared Exponential Kernel



A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$



Standard Kernels

Squared Exponential Kernel

A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

Neil Lawrence says that this kernel should be called the "Exponentiated Quadra

- The lengthscale ℓ determines the length of the 'wiggles' in your function. I
- The output variance σ^2 determines the average distance of your function

Rational Quadratic Kernel

$$k_{\mathrm{RQ}}(x, x') = \sigma^2 \left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

This kernel is equivalent to adding together many SE kernels with different leng

Pitfalls of the SE and RQ kernels

Most people who set up a GP regression or classification model end up using th derivatives (for example, the abs() function), then either your lengthscale will er extrapolate in smooth regions if there is even a small non-smooth region in you

If your data is more than two-dimensional, it may be hard to detect this proble

Periodic Kernel

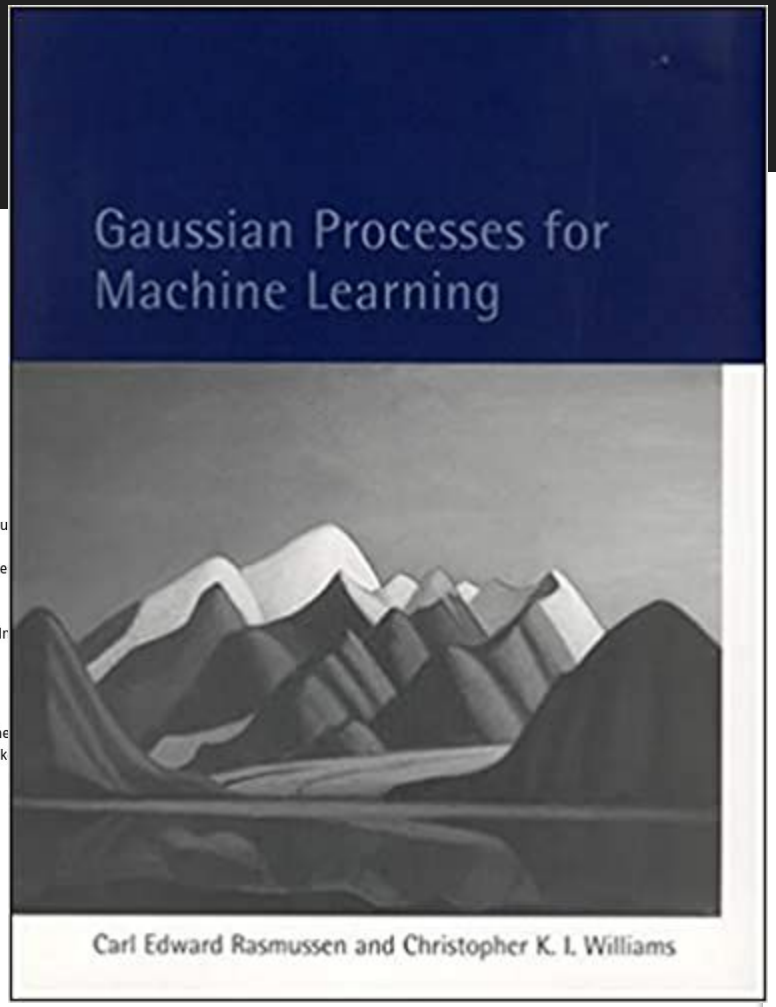
$$k_{\mathrm{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right)$$

The periodic kernel (derived by David Mackay) allows one to model functions wi

- The period p simply determines the distnace between repititions of the fu
- The lengthscale ℓ determines the lengthscale function in the same way as

Locally Periodic Kernel

Kernels



If you've ever asked yourself: "How do I choose the covariance function for a Gau

If you're looking for software to implement Gaussian process models, I recomme
they include a sensible default?"

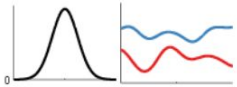
The answer is that the choice of kernel (a.k.a. covariance function) determines alr
about kernels to choose a sensible one, read on.

Support Vector Machines

If your question is: "How do I choose a kernel for a Support Vector Machine"? The
hard time learning all the parameters by cross-validation. This is why most SVM k

Standard Kernels

Squared Exponential Kernel

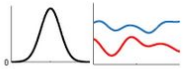


A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

Standard Kernels

Squared Exponential Kernel



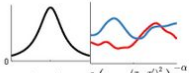
A.K.A. the Radial Basis Function kernel, the Gaussian kernel. It has the form:

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

Neil Lawrence says that this kernel should be called the "Exponentiated Quadra

- The lengthscale ℓ determines the length of the 'wiggles' in your function. I
- The output variance σ^2 determines the average distance of your function

Rational Quadratic Kernel



$$k_{\mathrm{RQ}}(x, x') = \sigma^2 \left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$$

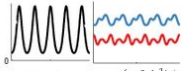
This kernel is equivalent to adding together many SE kernels with different leng

Pitfalls of the SE and RQ kernels

Most people who set up a GP regression or classification model end up using th
derivatives (for example, the abs() function), then either your lengthscale will er
extrapolate in smooth regions if there is even a small non-smooth region in you

If your data is more than two-dimensional, it may be hard to detect this proble

Periodic Kernel



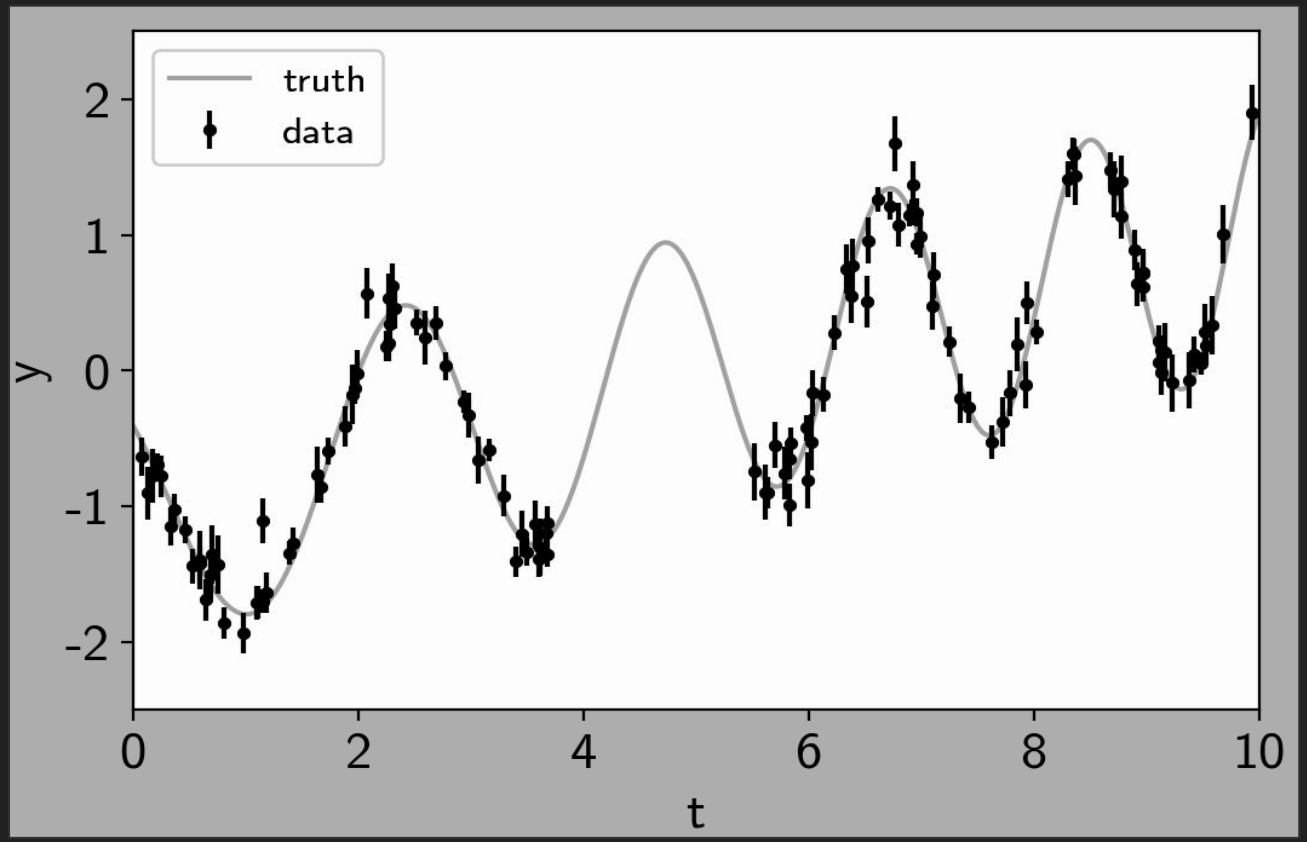
$$k_{\mathrm{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right)$$

The periodic kernel (derived by David Mackay) allows one to model functions w

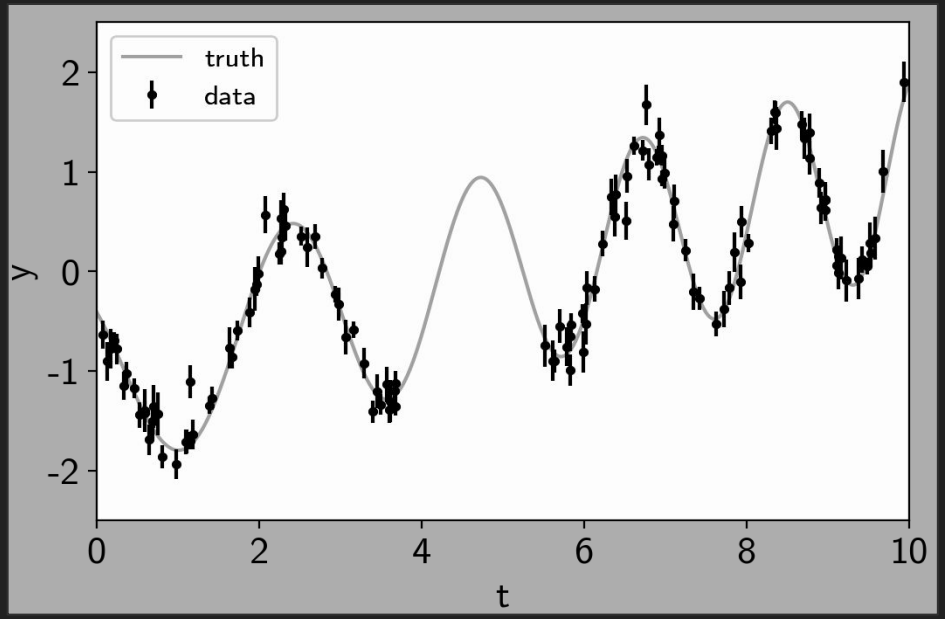
- The period p simply determines the distnace between repititions of the fu
- The lengthscale ℓ determines the lengthscale function in the same way as

Locally Periodic Kernel

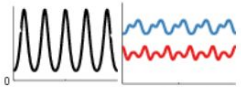
What can we do with GP?



What can we do with GP?



Periodic Kernel

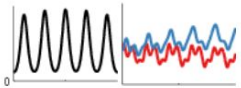


$$k_{\text{Per}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right)$$

The periodic kernel (derived by David Mackay) allows one to model functions which repeat themselves ex

- The period p simply determines the distnace between repititions of the function.
- The lengthscale ℓ determines the lengthscale function in the same way as in the SE kernel.

Locally Periodic Kernel



A SE kernel times a periodic results in functions which are periodic, but which can slowly vary over time.

$$k_{\text{LocalPer}}(x, x') = k_{\text{Per}}(x, x') k_{\text{SE}}(x, x') = \sigma^2 \exp\left(-\frac{2\sin^2(\pi|x-x'|/p)}{\ell^2}\right) \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

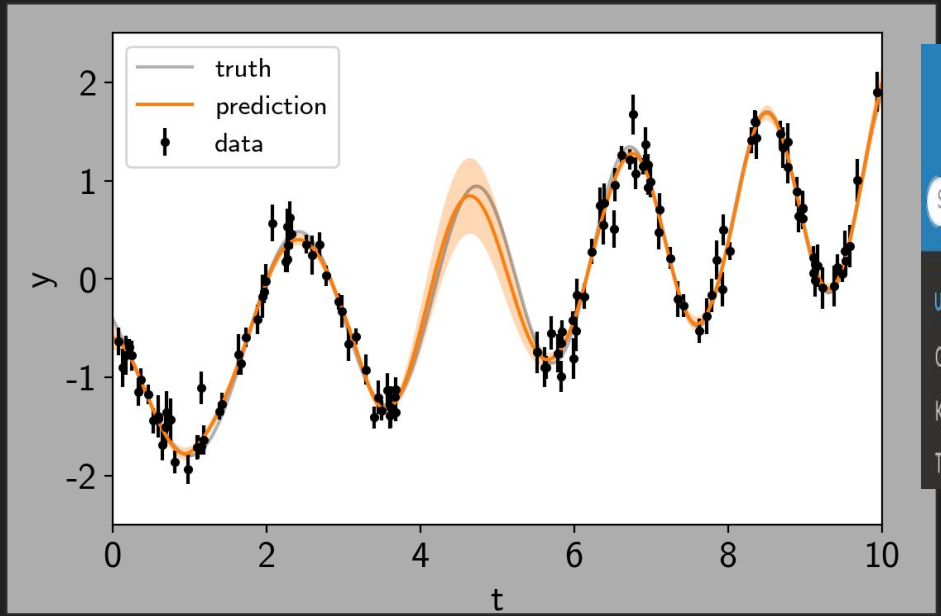
Most periodic functions don't repeat themselves exactly. To add some flexibility to our model, we can cor
shape of the repeating part of the function can now change over time.

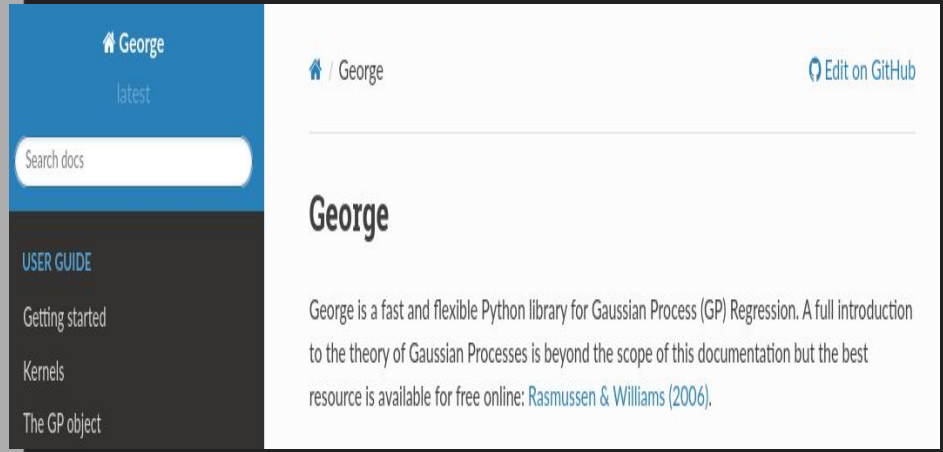
SHO kernel:

$$S(\omega) = \sqrt{\frac{2}{\pi}} \frac{S_1 {\omega_1}^4}{(\omega^2 - {\omega_1}^2)^2 + 2\,{\omega_1}^2\,\omega^2} + \sqrt{\frac{2}{\pi}} \frac{S_2 {\omega_2}^4}{(\omega^2 - {\omega_2}^2)^2 + {\omega_2}^2\,\omega^2/Q^2}$$

What can we do with GP?

```
with model:
    pred = gp.predict(true_t, return_var=True)
    for i, sample in enumerate(get_samples_from_trace(trace, size=N_pred)):
        pred_mu[i], pred_var[i] = eval_in_model(pred, sample)
```





How is Deep Learning different from classical methods?

Deep Learning

-Linearity not required

-Covariance of data not necessary apriori

-Scalable to computational limits

-Black box

-Trained

Classical

-Knowledge of linearity required

-Knowledge of covariance required (i.e an equation)

-Easy to understand

-Only data of interest required

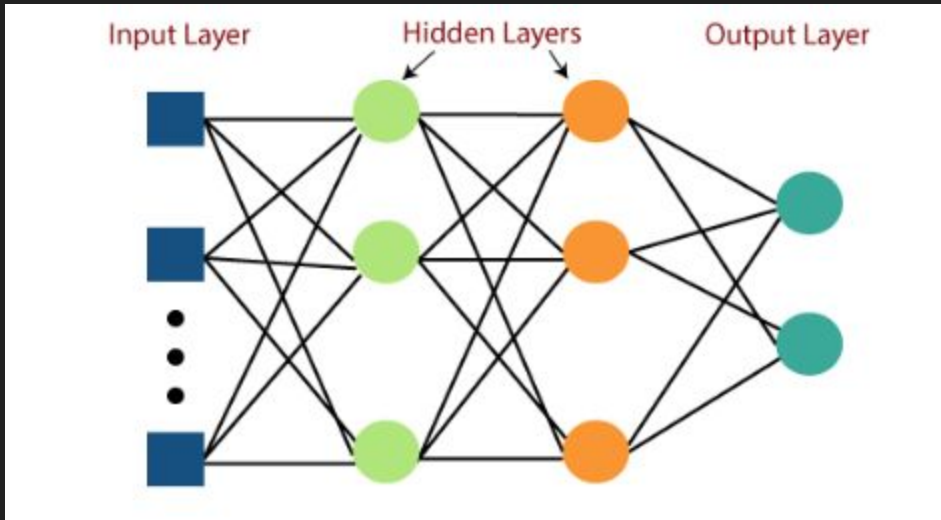
Deep learning is just fitting without the structure of a fitted EQ -

We do the exact same task as classical methods

Instead of fitting to F(x) we create a system which acts as a generalisation for all possible F(x).

AKA

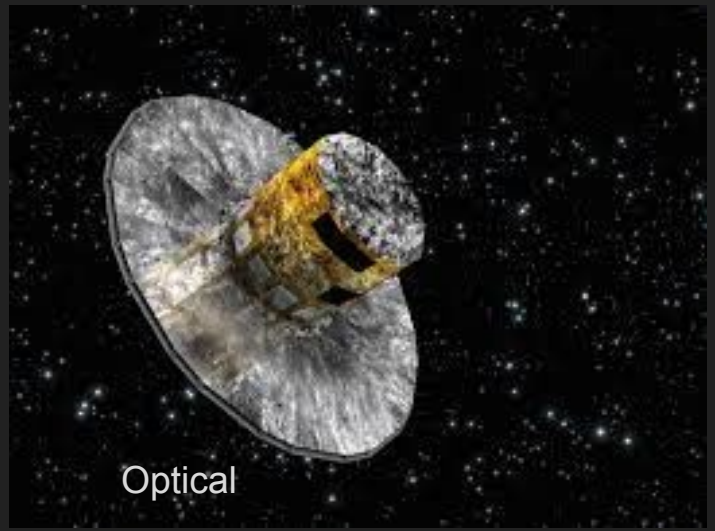
Universal Function Approximators

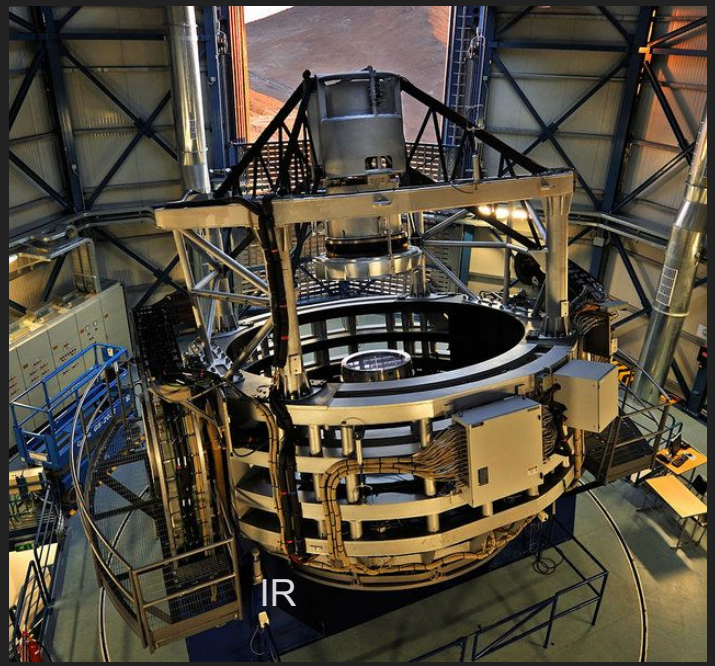


Regression - Improving parallax - *A very basic example using VVV*

Gaia is very good at parallax but sometime in space there is dust

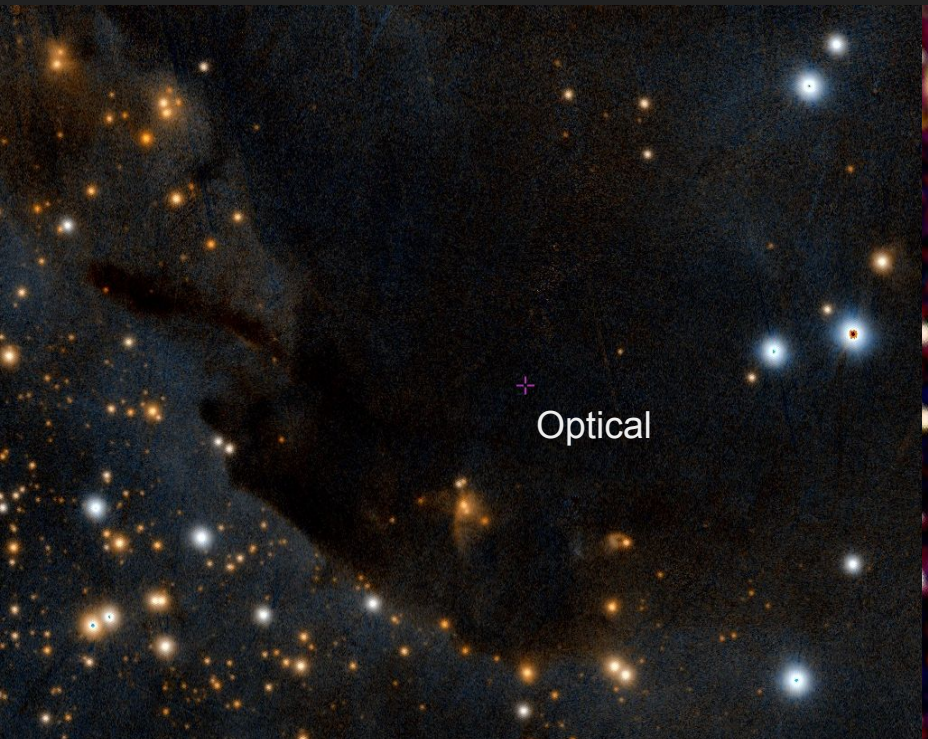
Optical light is typically blocked by dust :(

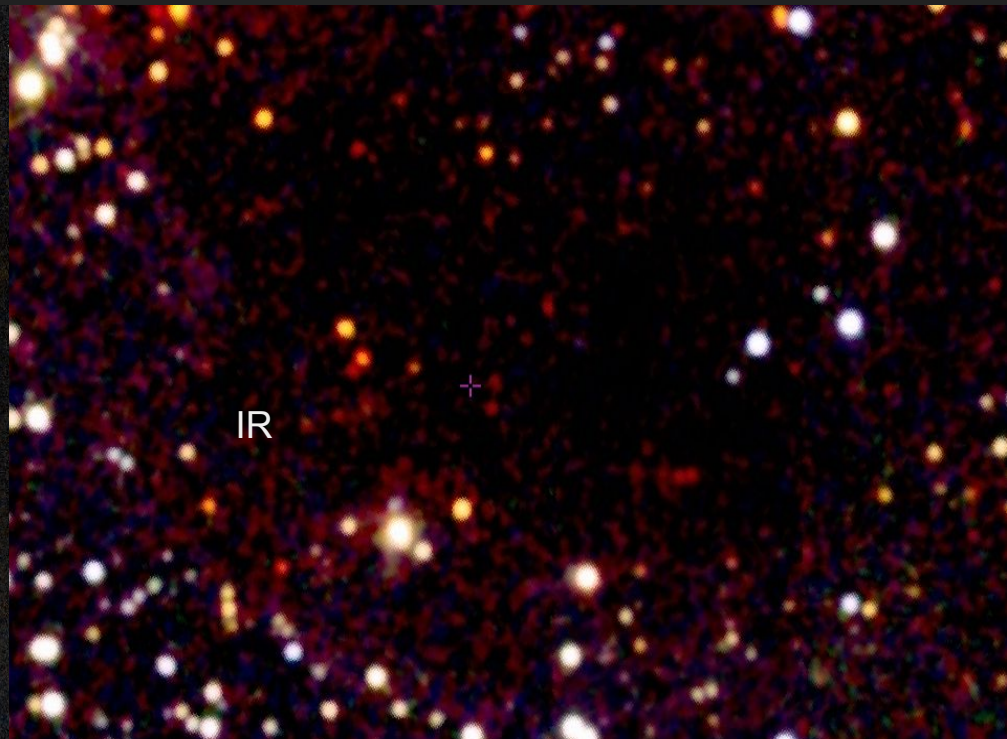




Regression - Improving parallax - *A very basic example using VVV*

Gaia is very good at parallax but sometime in space there is dust





A multilayer perceptron (MLP) - likely your first NN

- From 1958!!!
- The classic “Neural Network”

Input is X and Y

Y = Value we want and also know in some cases

X = Other values which correlate with Y

```python
class MLP(nn.Module):
    def __init__(self, n_inputs: int):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(n_inputs, 128)
        self.bn1 = nn.BatchNorm1d(128)
        self.act1 = nn.ReLU()

        self.hidden2 = nn.Linear(128, 128)
        self.bn2 = nn.BatchNorm1d(128)
        self.act2 = nn.ReLU()

        self.hidden3 = nn.Linear(128, 64)
        self.bn3 = nn.BatchNorm1d(64)
        self.act3 = nn.ReLU()

        self.hidden4 = nn.Linear(64, 32)
        self.bn4 = nn.BatchNorm1d(32)
        self.act4 = nn.ReLU()

        self.hidden5 = nn.Linear(32, 1)

        self._initialize_weights()
```

A multilayer perceptron (MLP) - likely your first NN

I think any combination of these values :

```
x_filter = ['parallax_corr','phot_bp_rp_excess_factor_corr',
    'ra','dec','l','b','ecl_lon','ecl_lat',
    'parallax','pmra','pmdec',
    'dec_parallax_corr','dec_pmdec_corr','dec_pmra_corr',
    'parallax_pmdec_corr','parallax_pmra_corr',
    'pm','pmra_pmdec_corr','ra_dec_corr','radial_velocity',
    'ra_parallax_corr','ra_pmdec_corr','ra_pmra_corr',
    'ra_vvv','dec_vvv','l_vvv','b_vvv','parallax_vvv',
    'pmra_vvv','pmdec_vvv',
    'bp_g','bp_rp',
    'g_rp','grvs_mag',
    'J-K','H-K','Z-K','Y-K']
```

Could represent this value:

```
y_filter = ['parallax_corr']
```

A multilayer perceptron (MLP) - likely your first NN

For each example of X and Y I have…

Given X, what does the MLP think Y is?

As we know, loss = how wrong it was

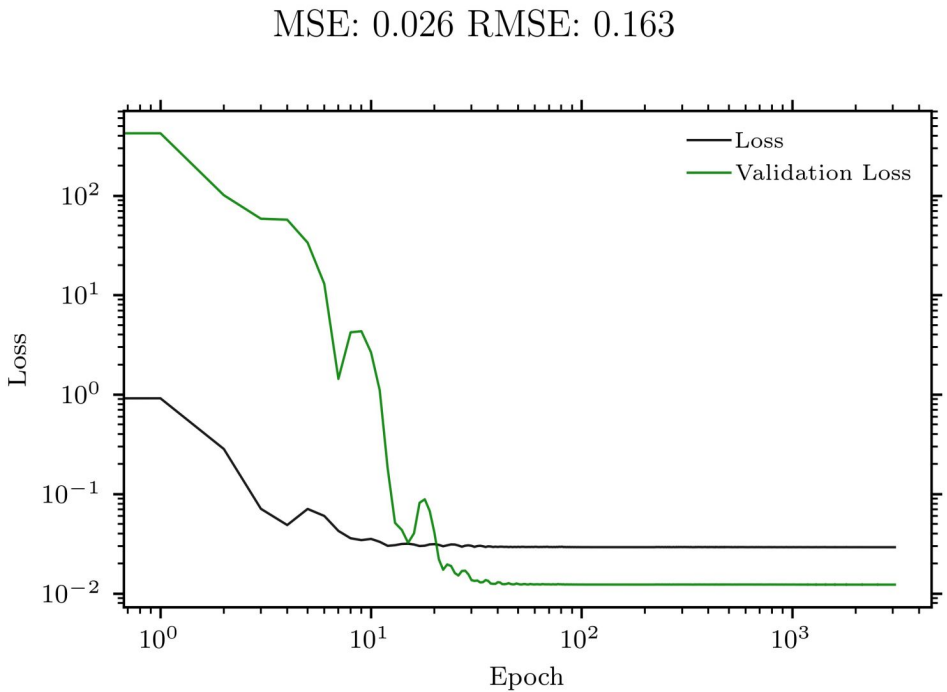
We can use that to calculate the direction

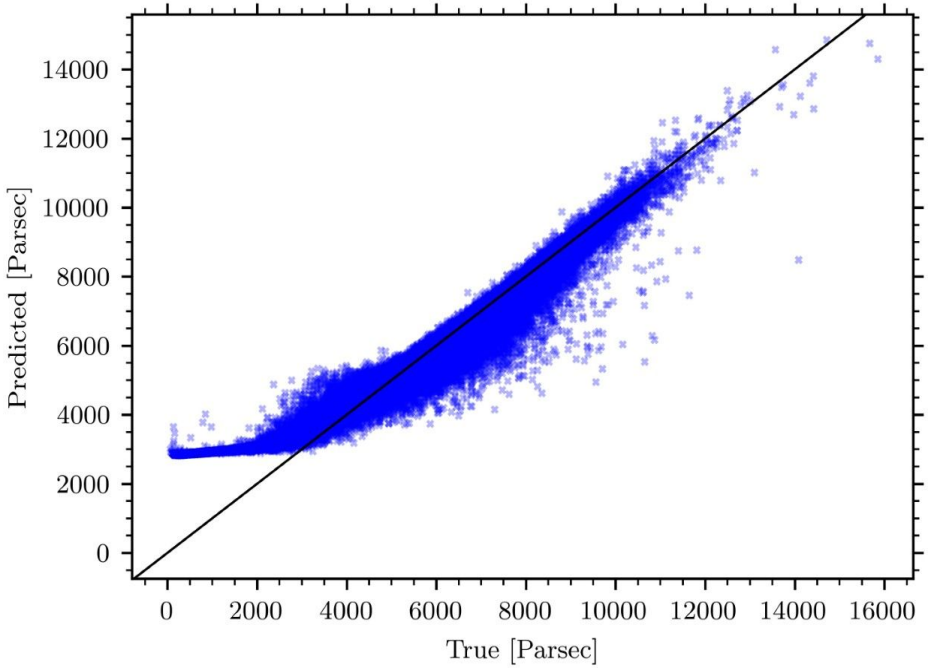
Inform the network on how wrong it was

```
for epoch in range(epochs):
    model.train()
    for train_features, train_labels in train_dl:
        opt.zero_grad()
        yhat = model(train_features)
        loss = criterion(yhat, train_labels)
        loss.backward()
        opt.step()
```

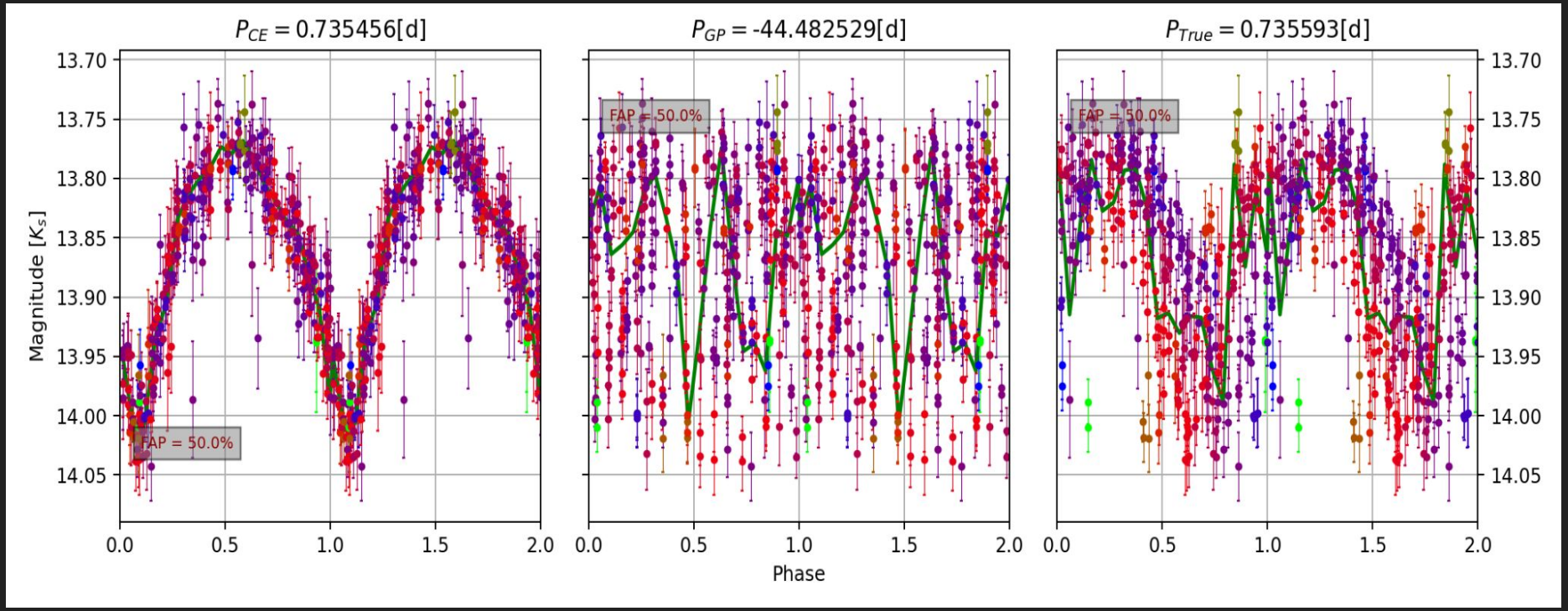
(à la the prior gradient descent)

And what do we get?

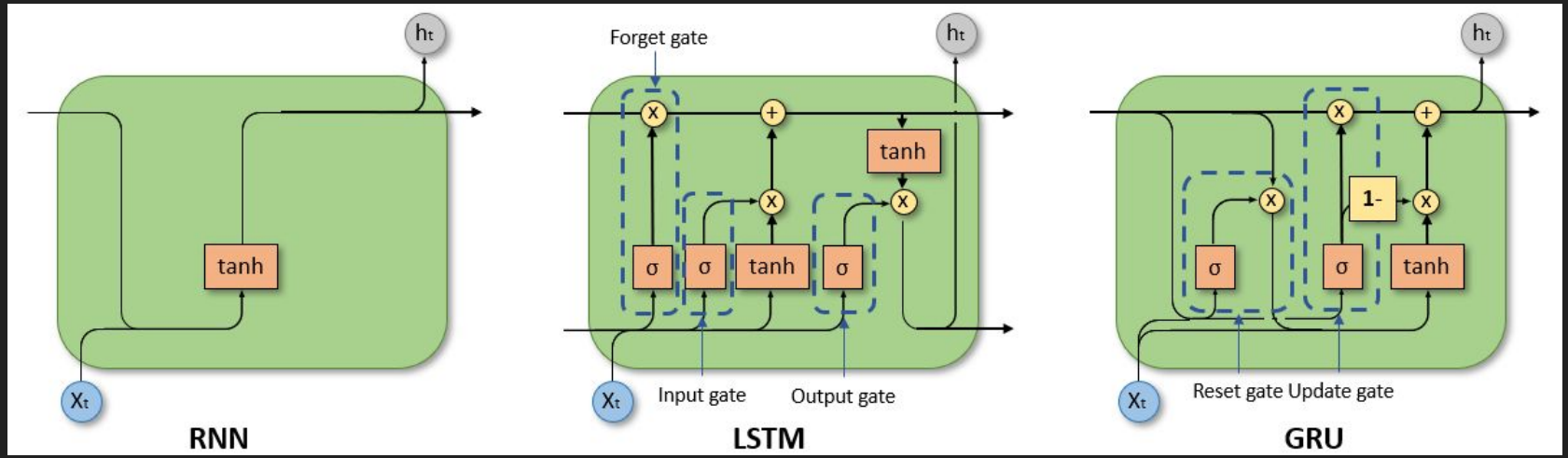




Same star, 3 different periods, which is correct?



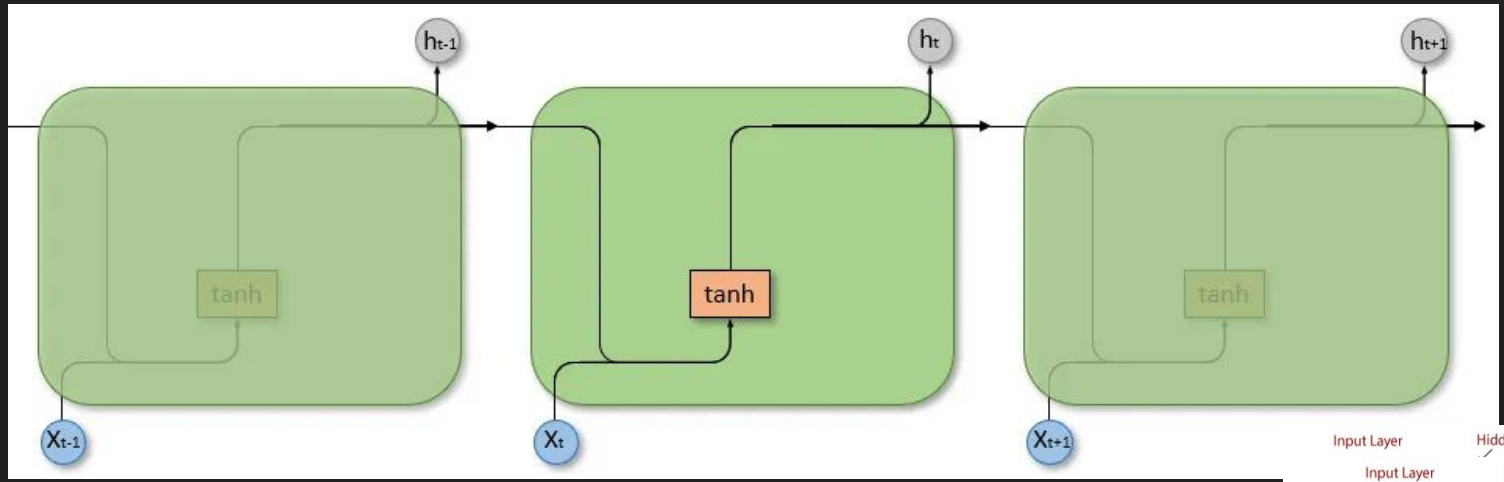
Light curves are not collections of features…



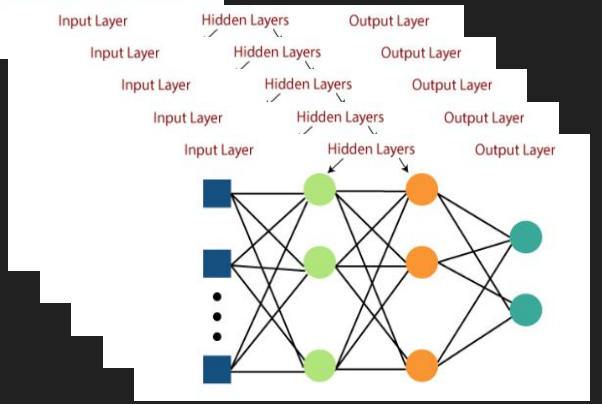
Recurrent Neural Network (RNN)
Long Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)

Or just an MLP with memory…

Light curves are not collections of features…



If we unroll a Recurrent Neural Network



Period finder and other stuff!

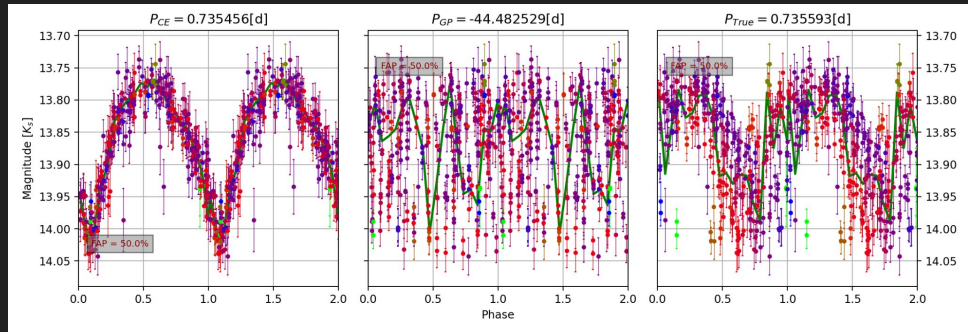
How do we know if a star is periodic?
How do we know if we have extracted the right period?

Manual inspection can't be the only way?!?

Checking 200 took me ~ 1hr, ∴ all 10^7 will take ~ 7 years

- Analyse the periodogram?
- Compare against other periodograms?
- Analyse the folded LC?

Same star, 3 different periods, which is correct?



Recurrent Neural Network (RNN) - likely your second NN

For each example of X and Y I have…

Given X, what does the RNN think Y is?

As we know, loss = how wrong it was

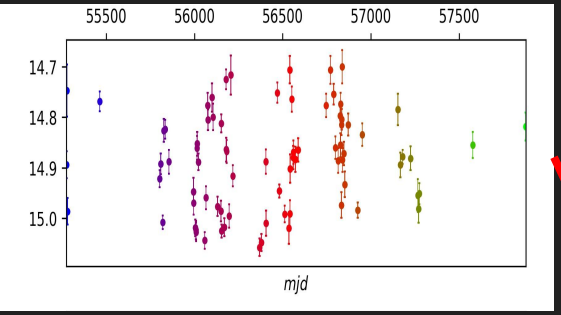
We can use that to calculate the direction

Inform the network on how wrong it was

```
for epoch in range(epochs):
    model.train()
    for train_features, train_labels in train_dl:
        opt.zero_grad()
        yhat = model(train_features)
        loss = criterion(yhat, train_labels)
        loss.backward()
        opt.step()
```

(à la the prior gradient descent)

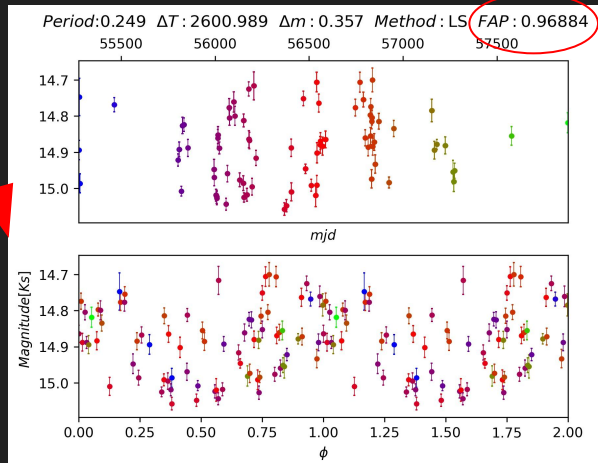
Hello AI, Is this star periodic?



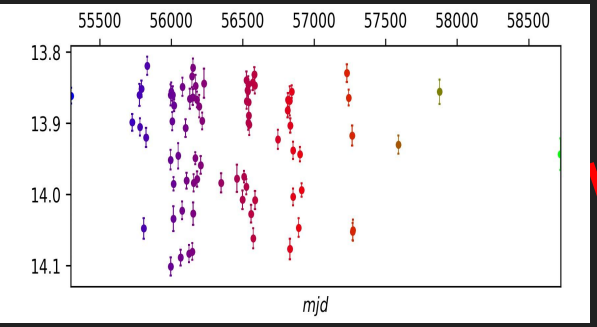
Big **F**alse **A**larm **P**robability

96.9% chance it's NOT periodic

NO



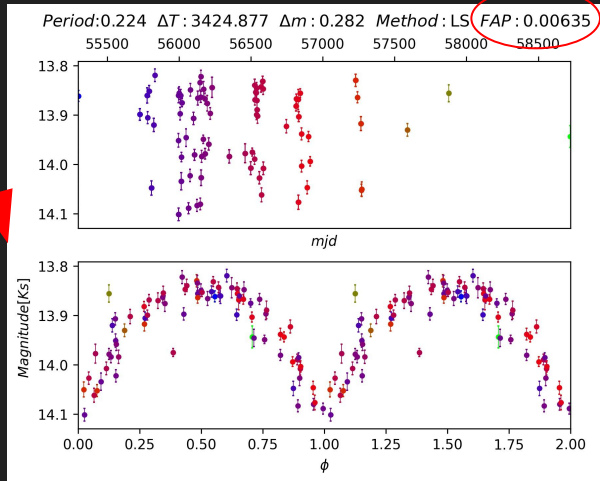
Hello AI, Is this star periodic?



Small **F**alse **A**larm **P**robability

0.6% chance it's NOT periodic
Or
99.4% chance it is periodic

YES :)



The verification of periodicity with the use of recurrent neural networks

N. Miller*, P. W. Lucas[1], Y. Sun[2]
[1]*Centre for Astrophysics Research, University of Hertfordshire, College Lane, Hatfield, Hertfordshire, AL10 9AB, UK*
[2]*Centre for Computer Science and Robotics Research, University of Hertfordshire, College Lane, Hatfield, Hertfordshire, AL10 9AB, UK*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

The ability to automatically and robustly self-verify periodicity present in time-series astronomical data is becoming more important as data sets rapidly increase in size. The age of large astronomical surveys has rendered manual inspection of time-series data less practical. Previous efforts in generating a false alarm probability to verify the periodicity of stars have been aimed towards the analysis of a constructed periodogram. However, these methods feature correlations with features that do not pertain to periodicity, such as light curve shape, slow trends and stochastic variability. The common assumption that photometric errors are Gaussian and well determined also a limitation of analytic methods. We present a novel machine learning based technique which directly analyses the phase folded light curve for its false alarm probability. We show that results of this method are largely insensitive to the shape of the light curve, and we establish minimum values for the number of data points and the amplitude to noise ratio.

Key words:
stars: variables: general – methods: observational – methods: statistical

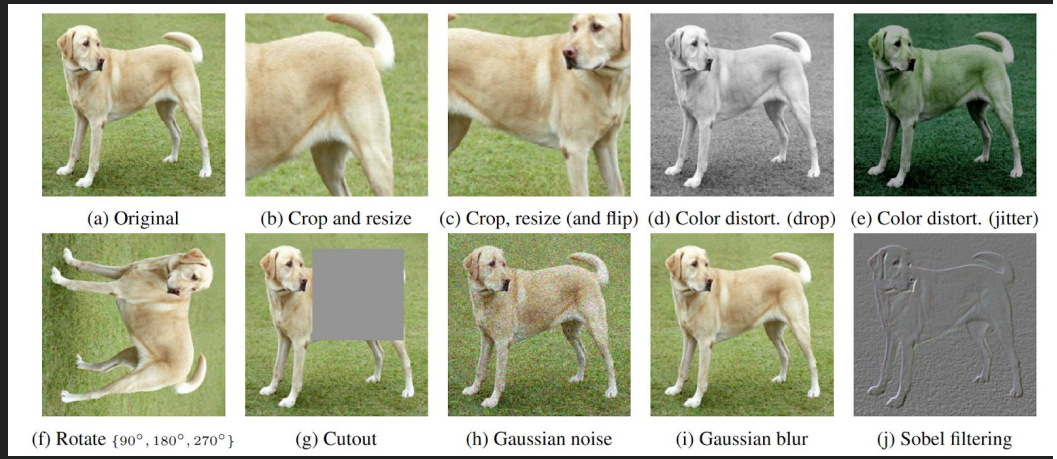
1 INTRODUCTION

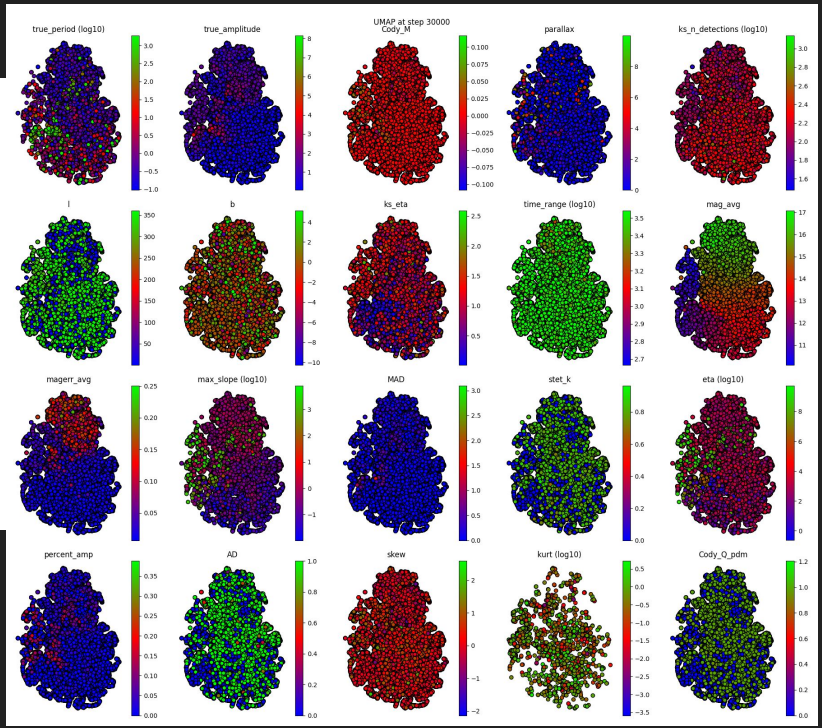
The identification of periodic variable stars is not a trivial task; well-understood statistical measures can be used to identify variability in time-series but not so easily periodic variability. The Stetson variability index '*J*' (Stetson 1996) compares the variability of each

2010; Borucki et al. 2003; Ivezić et al. 2019; Ricker et al. 2015) we anticipate time-series catalogs of sizes that render sufficient manual inspection increasingly non-viable. Hence, a reliable and robust metric for identifying periodicity is required.

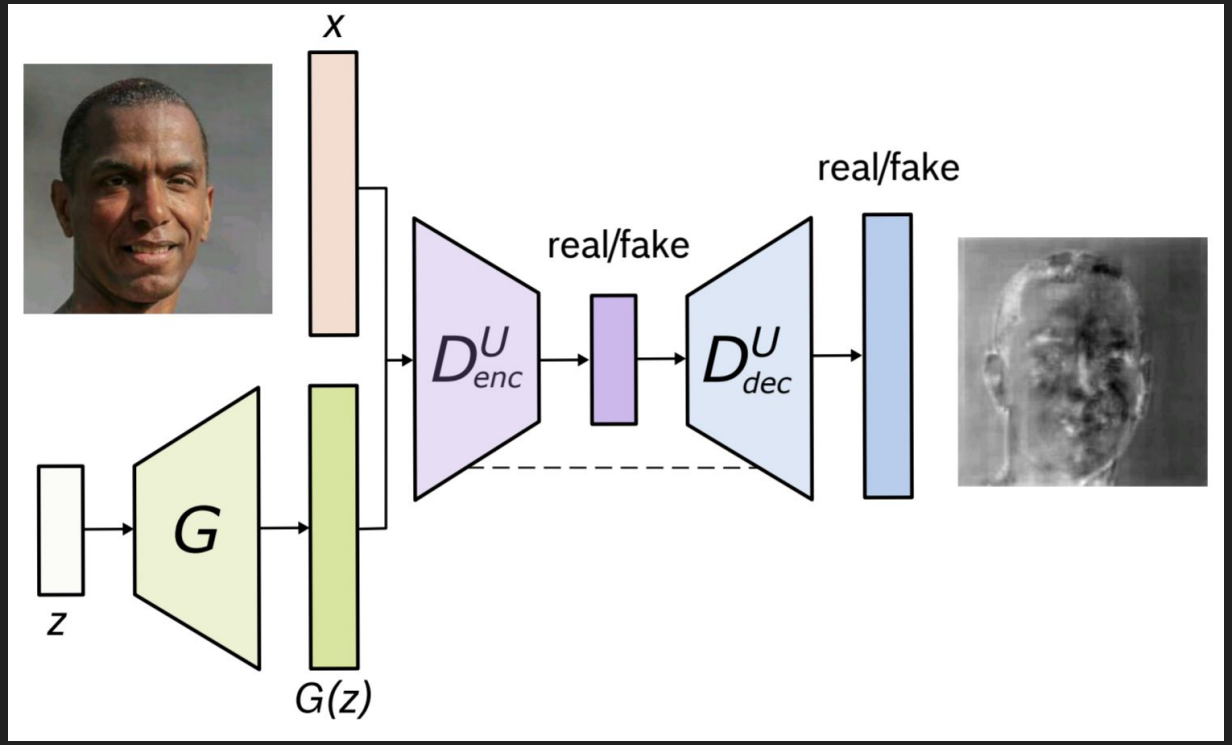
It is not a guarantee that a large survey will feature high cadence

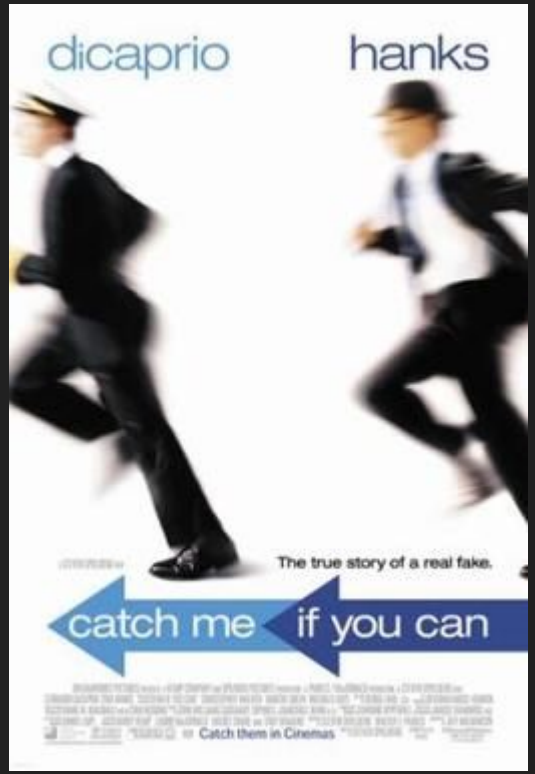
Classification - classification of unknown unknowns



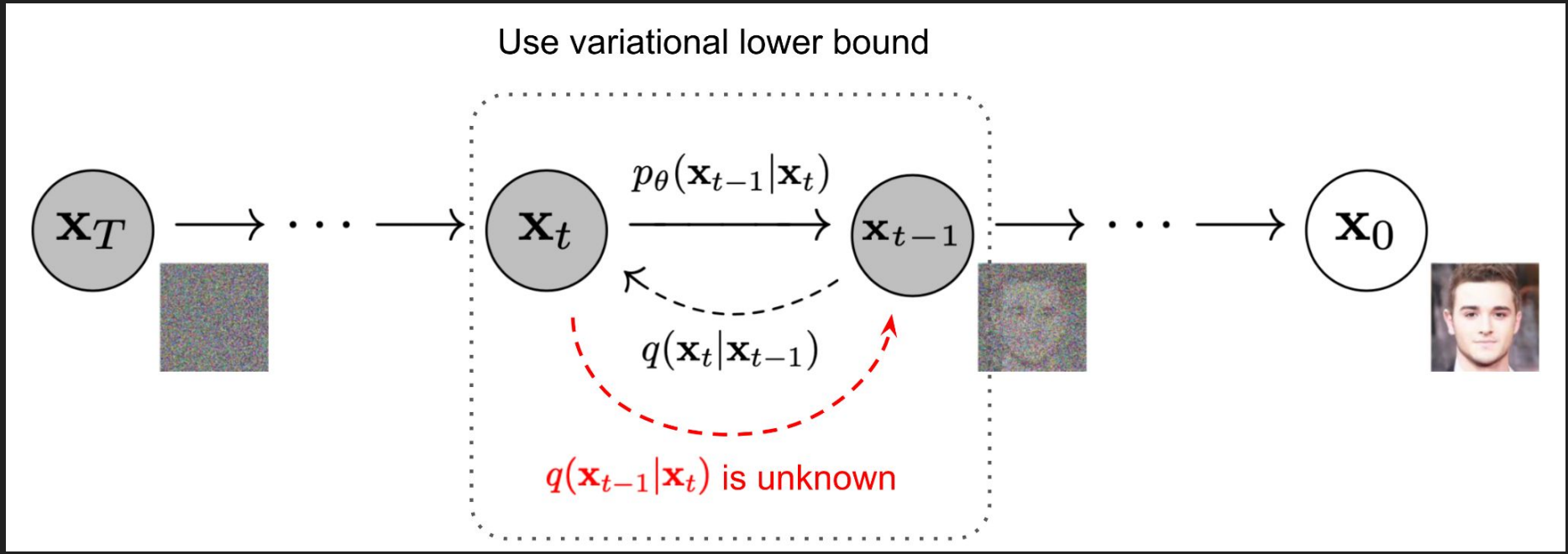


Fake light curves

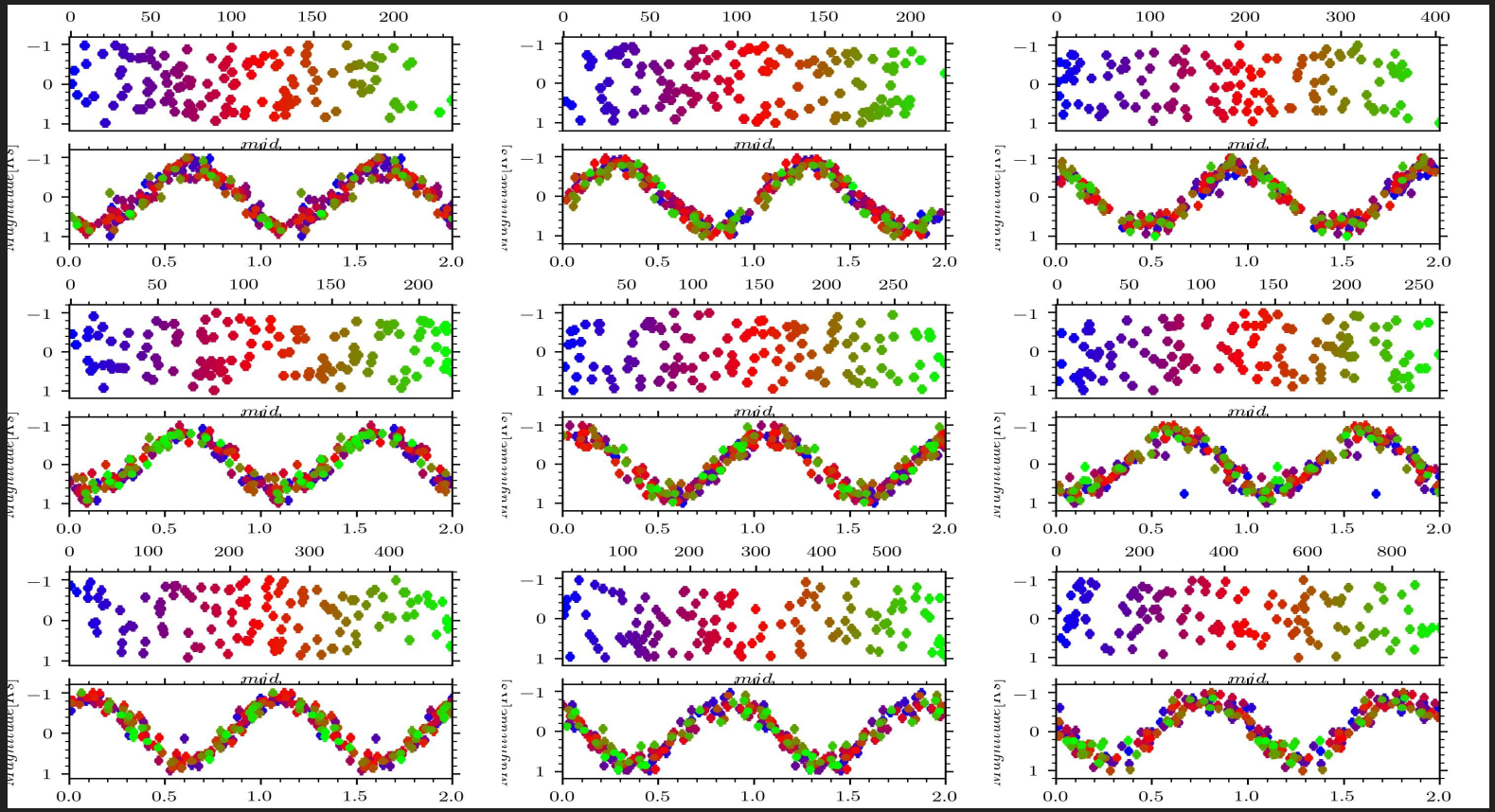




Fake light curves



Fake light curves



Extra stuff

How to reliably extract info

What we don't know:

- Light Curve shape
- Light Curve photometric accuracy
- Light Curve contamination
- Other sources of perturbation

What we know:

- Hopefully a star?
- Probably variable

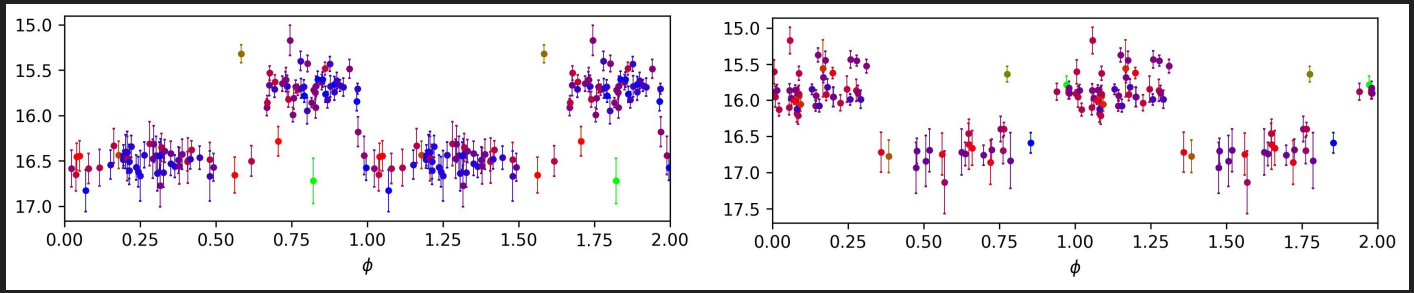
How to reliably extract info

What we don't know:

- Light Curve shape
- Light Curve photometric accuracy
- Light Curve contamination
- Other sources of perturbation

What we know:

- Hopefully a star?
- Probably variable



Lomb Scargle

- By far the most common
- Fourier based
- Is a fitting technique appropriate?
- Why bother using it?
  - Its fast
  - Something of an “industry standard”
  - My L-S should agree with your L-S

Understanding the Lomb–Scargle Periodogram

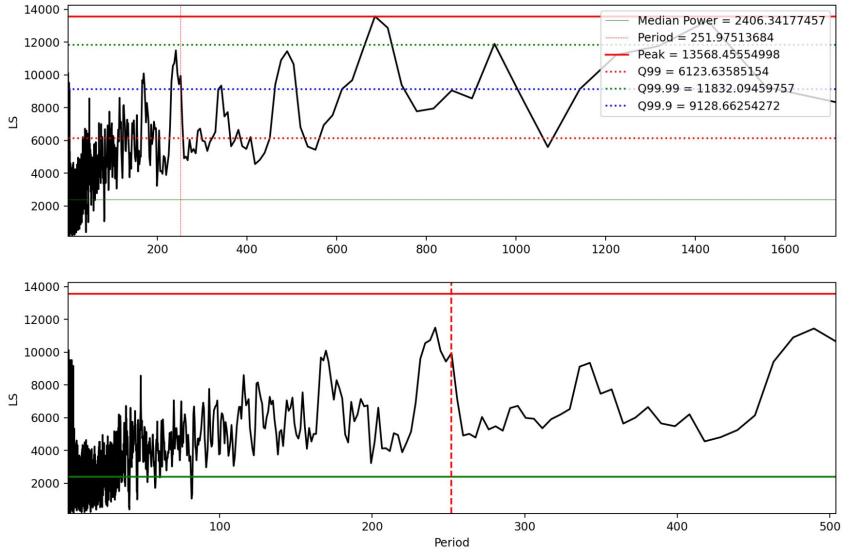
Jacob T. VanderPlas
University of Washington, eScience Institute, 3910 15th Ave NE, Seattle WA 98195
Received 2017 August 26; revised 2017 December 6; accepted 2017 December 7; published 2018 May 11

LEAST-SQUARES FREQUENCY ANALYSIS OF UNEQUALLY SPACED DATA

N. R. LOMB
School of Physics, University of Sydney, N.S.W., Australia

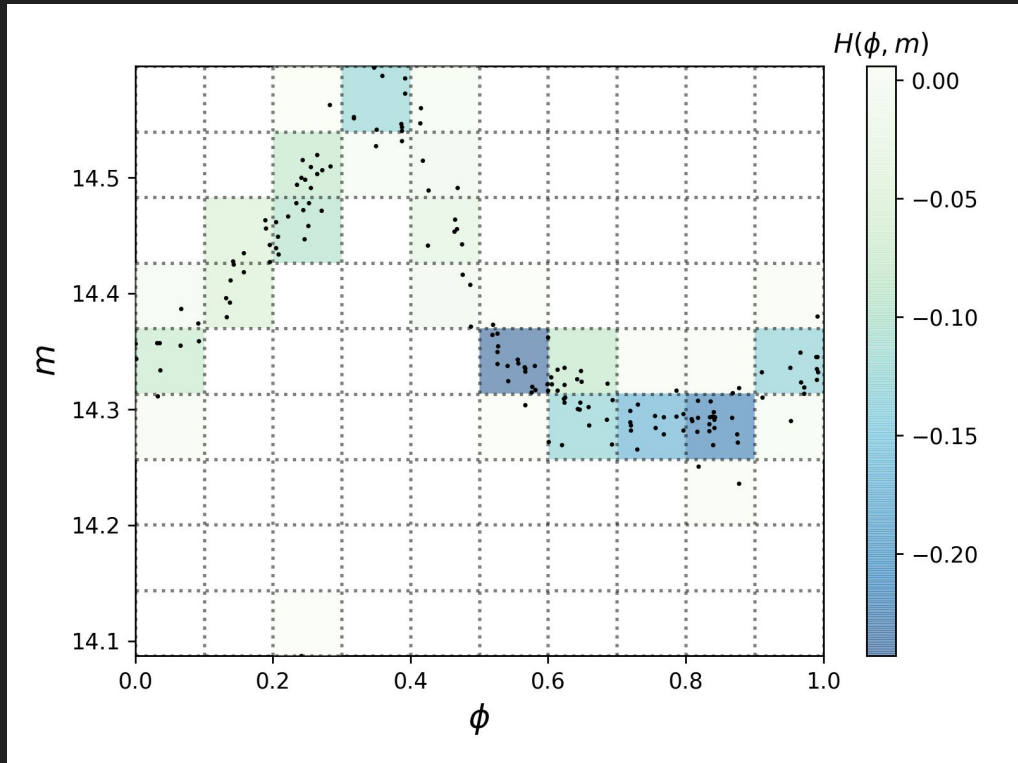
STUDIES IN ASTRONOMICAL TIME SERIES ANALYSIS. II. STATISTICAL ASPECTS OF SPECTRAL ANALYSIS OF UNEVENLY SPACED DATA

JEFFREY D. SCARGLE
Theoretical and Planetary Studies Branch, Space Science Division, Ames Research Center, NASA
Received 1982 January 11; accepted 1982 April 26



Phase Dispersion Minimisation & Conditional Entropy

- Phase folding techniques
- Simply finds ‘cleanest’ phase fold
- Nothing assumed about the structure
- Computationally expensive
- Not practical to perfectly tune



PERIOD DETERMINATION USING PHASE DISPERSION MINIMIZATION

R. F. STELLINGWERF

Department of Physics and Astronomy, Rutgers University

Received 1978 February 6; accepted 1978 March 22

Using conditional entropy to identify periodicity

Matthew J. Graham,* Andrew J. Drake, S. G. Djorgovski, Ashish A. Mahabal and Ciro Donalek

California Institute of Technology, 1200 E. California Blvd, Pasadena, CA 91125, USA

2013

35 years between them!!!