Imperial-X

Machine Learning Basics for ~~Plasma Physics~~ Regression and Inference

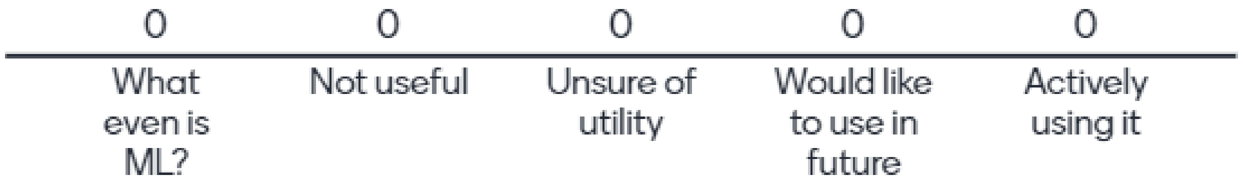
Dr Aidan Crilly

Eric and Wendy Schmidt AI in Science Postdoctoral Fellow

Member of Plasma Group within the Community of Natural Environment

Opinion on machine learning in your research:

*Go to menti.com and enter code ** ** ** **



What to expect?

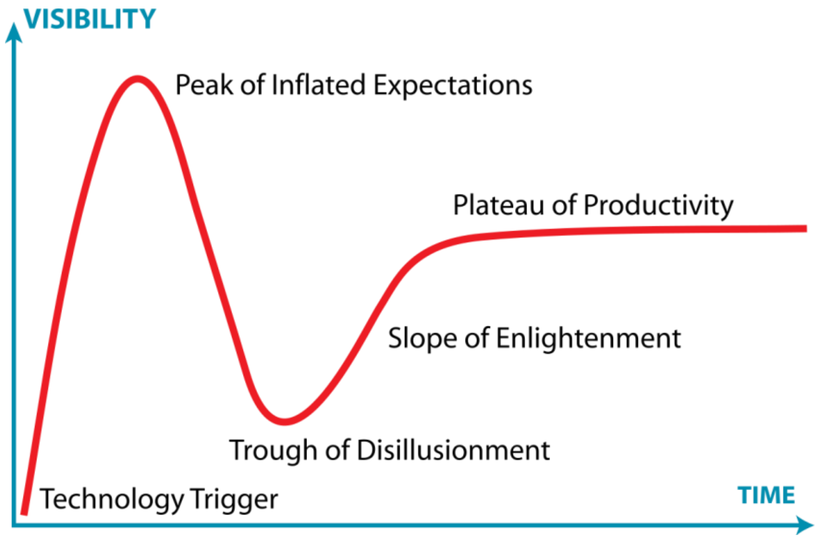
- This talk aims to:
  - Introduce key machine learning (ML) concepts and language
  - Relate ML techniques to more familiar numerical methods
  - Introduce ML techniques which have been used in Plasma Group's research
  - Hopefully, be a jumping off point to learning more about ML useful to your own research
- This talk does not aim to:
  - Discuss ML research, we will discuss ML *in* research
  - Be an authority on ML, it would be great if this could become a discussion on use cases

Overview

- What is Machine Learning?
- Regression & Classification
- Parametric Regressors (Non-neural)
  - Ordinary and Non-Linear Least Squares
- Bayesian Inference
- Non-Parametric Regressors
  - Gaussian Processes
- Neural Networks

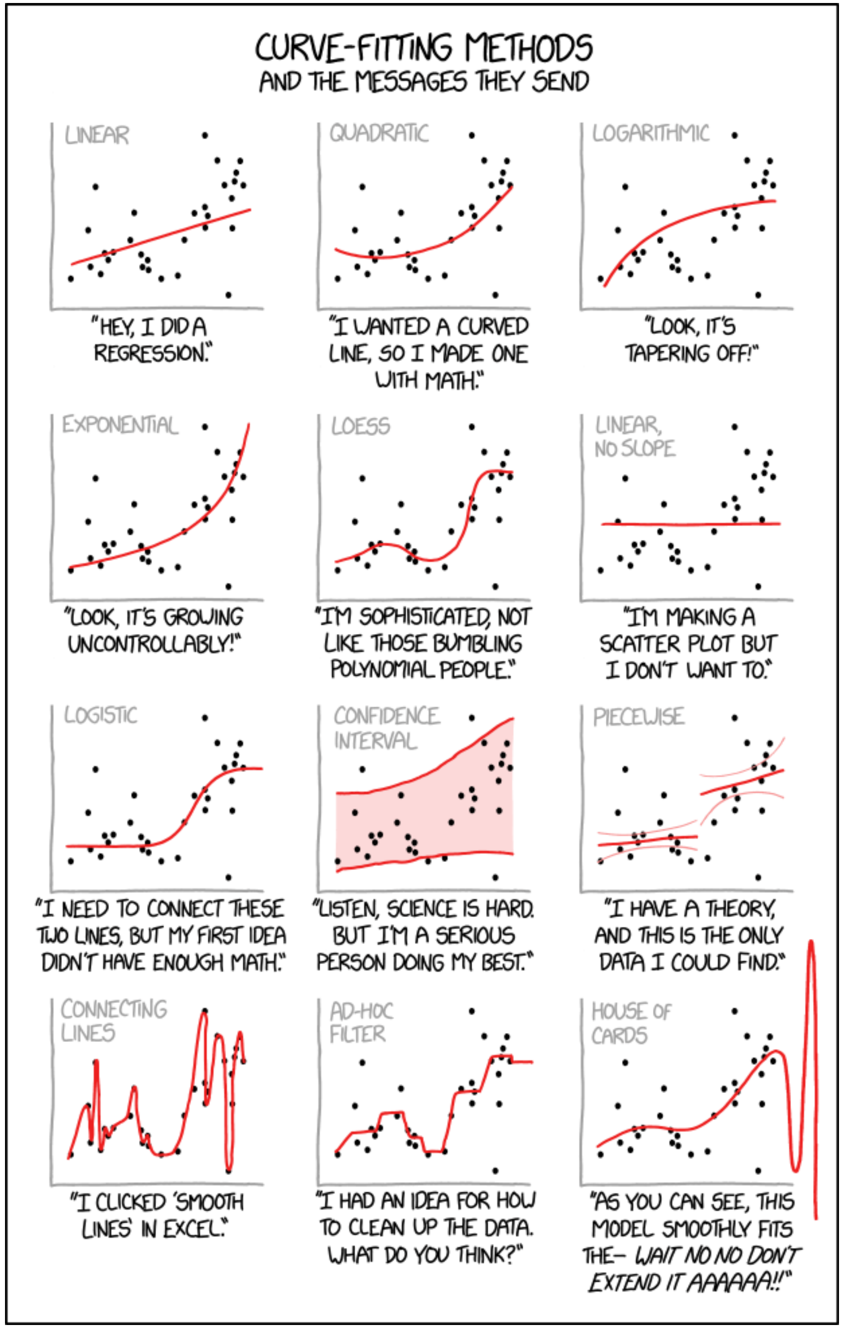
What is “Machine Learning”?

- Numerical modelling which adapts parameters algorithmically to learn the relationship between input and output data
- Parameter evolution often posed as an optimisation problem (nothing new!)
- User often must select ‘hyper-parameters’ which change aspects of the machine learning algorithm



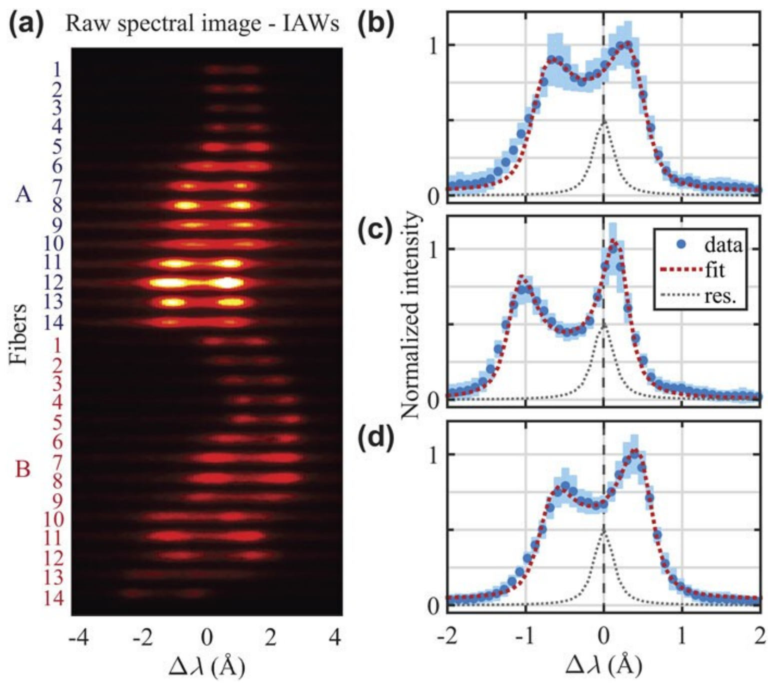
Regression & Classification

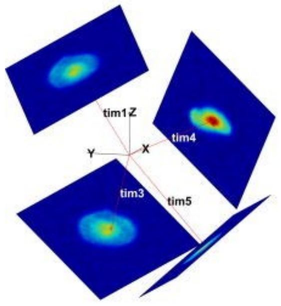
- Two of the core ML tasks are regression and classification
- Classification is the task of placing data within a finite number of sets, for example determining if a photo contains a cat or not
- *Regression is the task of estimating the functional relationship between a set of input and output variables, for example fitting a straight line to (x,y) data pairs*

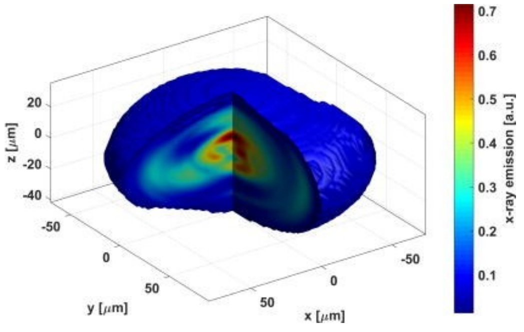


Regression

- Regression problems are ubiquitous in data analysis within the physical sciences
- An entirely non-exhaustive list of examples from my research:
  - Image reconstruction and 3D tomography
  - Thomson scattering spectra
  - Fusion neutron spectroscopy
  - X-ray line spectroscopy
- Typically, reduced models with small number of parameters are used to fit each diagnostic signal separately
- *Latent parameters of numerical models can also be learned via regression…*

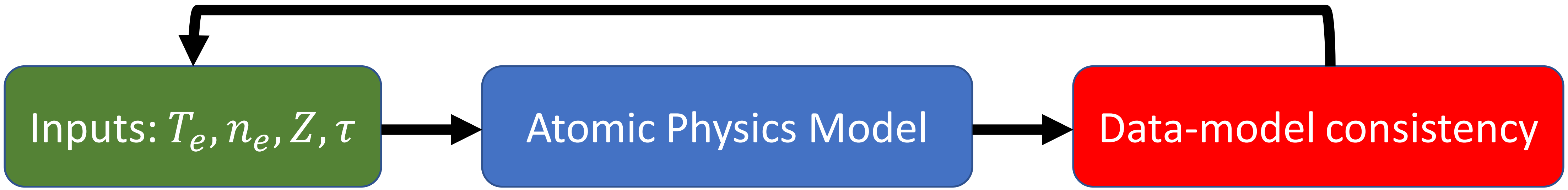






Parametric Regressors

- The most common form of regression includes a ‘physics-based’ model which maps input physical parameters (e.g. temperature and density) to the expected diagnostic signature
- For example, line spectroscopy



- The ‘optimal’ set of parameters which match the observed data are sought in a forward-fit process

Ordinary Least Squares

- Say we have a linear model with unknown parameters θ_j:

$$f(x, \theta) = \sum_j \theta_j f_j(x) = \underline{\boldsymbol{A}} \cdot \boldsymbol{\theta}$$

- And we wish to minimize the weighted least squares distance to the data y_i (we shall see why later):

$$\min_\theta \sum_i w_i [y_i - f(x_i, \boldsymbol{\theta})]^2 = \min_\theta \left(\boldsymbol{y}^T \underline{\boldsymbol{W}} \boldsymbol{y} + \boldsymbol{\theta}^{\boldsymbol{T}} \underline{\boldsymbol{A}}^T \underline{\boldsymbol{W}}\, \underline{\boldsymbol{A}} \boldsymbol{\theta} - 2 \boldsymbol{\theta}^{\boldsymbol{T}} \underline{\boldsymbol{A}}^T \underline{\boldsymbol{W}} \boldsymbol{y} \right)$$

$$\rightarrow \underline{\boldsymbol{A}}^T \underline{\boldsymbol{W}}\, \underline{\boldsymbol{A}}\, \widehat{\boldsymbol{\theta}} = \underline{\boldsymbol{A}}^T \underline{\boldsymbol{W}}\, \boldsymbol{y}$$

- Therefore, the ‘best fit’ is given by the solution to the ‘normal equations’:

$$\widehat{\boldsymbol{\theta}} = \left(\underline{\boldsymbol{A}}^T \underline{\boldsymbol{W}}\, \underline{\boldsymbol{A}} \right)^{-1} \underline{\boldsymbol{A}}^T \underline{\boldsymbol{W}}\, \boldsymbol{y}$$

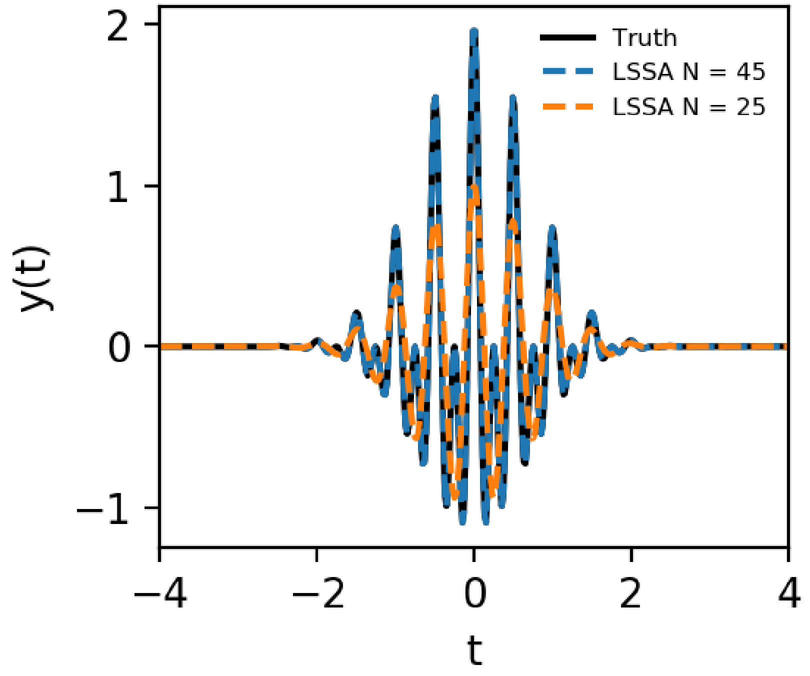
Practical example

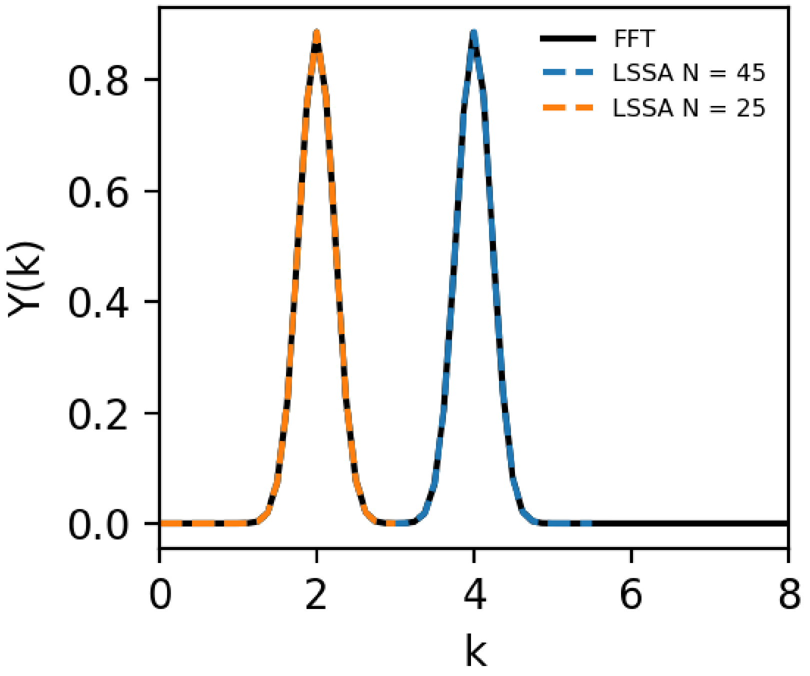
- Least-squares spectral analysis (LSSA) is a linear regression problem

$$f(x,\theta) = \sum_{j=0}^{N} \theta_j \cos\left[\frac{\pi j}{T} t\right] = \underline{\boldsymbol{A}} \cdot \boldsymbol{\theta}$$

$$\widehat{\boldsymbol{\theta}} = \left(\underline{\boldsymbol{A}}^T \underline{\boldsymbol{A}}\right)^{-1} \underline{\boldsymbol{A}}^T \boldsymbol{y}$$

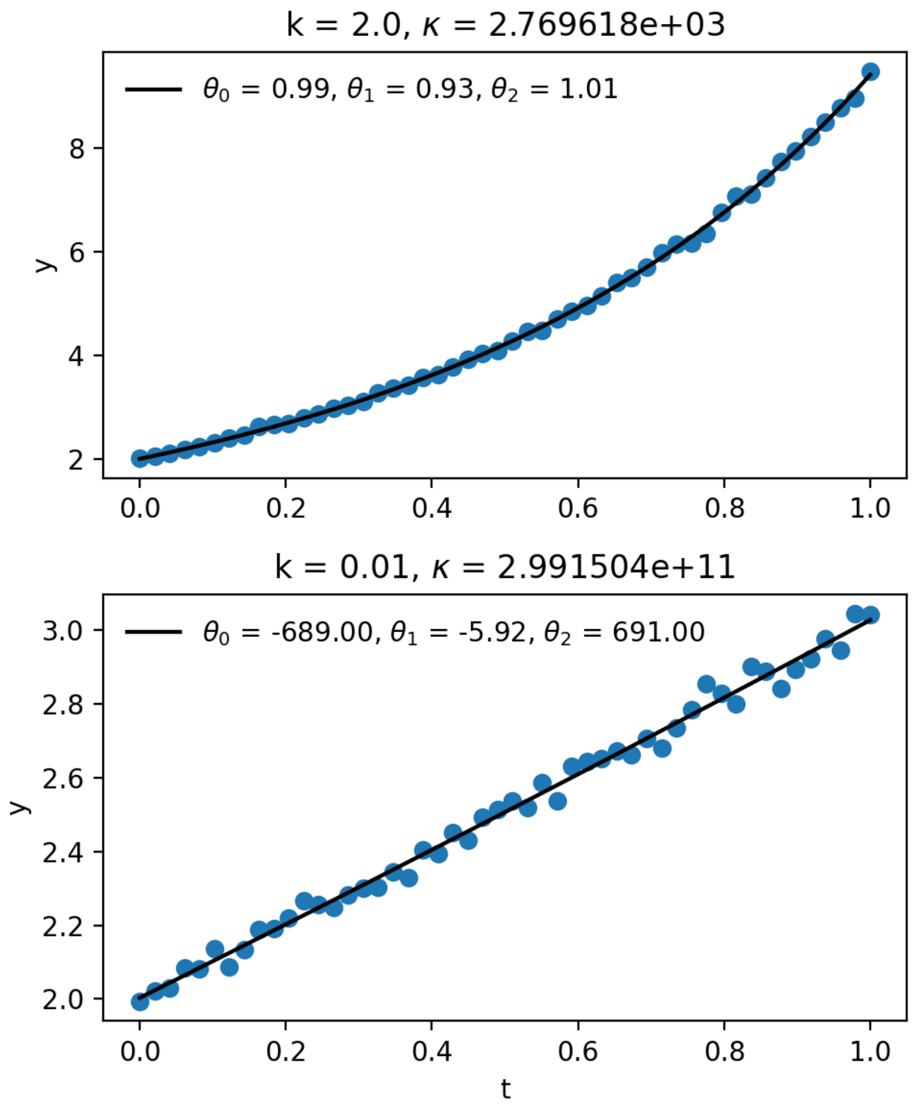
- Can handle unequally spaced data, uncertainties and limit the mode number
  - See Lomb-Scargle method
- Not formally a Discrete Fourier Transform





Ill-conditioned or ill-posed problems

- While linear systems have formal solutions they can be numerical unstable, small errors on one side of the equation can lead to large errors on the other
- One possible cause is degeneracy or collinearity of input parameters, e.g. fitting $f(\boldsymbol{\theta}, k, t) = \theta_0 + \theta_1 t + \theta_2 \exp[kt]$, for small kt
- These problems are ‘ill-conditioned’ as quantified by the condition number of the linear system



Regularisation

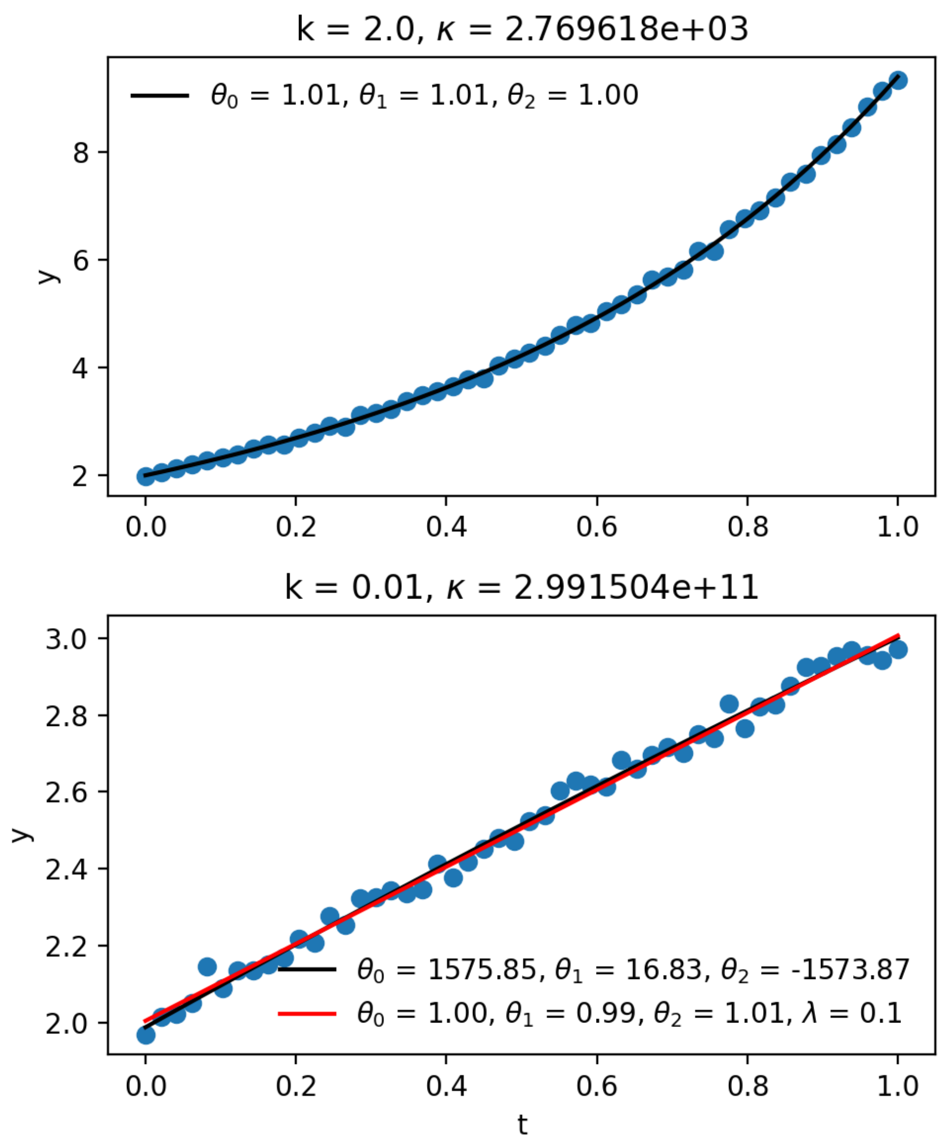
- Another key ML technique is regularisation
- It aims to reduce variance while introducing bias i.e. prevents overfitting

$$\mathrm{MSE} = \mathrm{Var}(f) + \mathrm{Bias}(f)^2 + \sigma^2$$

- Most commonly, Tikhonov regularisation

$$\min_{\theta} \sum_i [y_i - f(x_i, \boldsymbol{\theta})]^2 + \lambda |\boldsymbol{\theta}|^2$$

- λ is Lagrange multiplier on constraint $|\boldsymbol{\theta}|^2 = 0$
- Other methods include truncated SVD, spectral filtering, parameter priors, and many more



Deconvolution and Regularisation

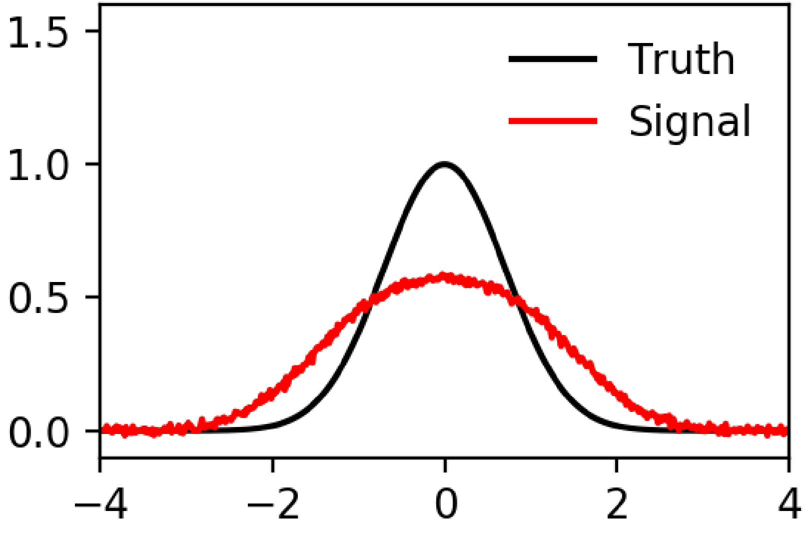
- Convolution is linear

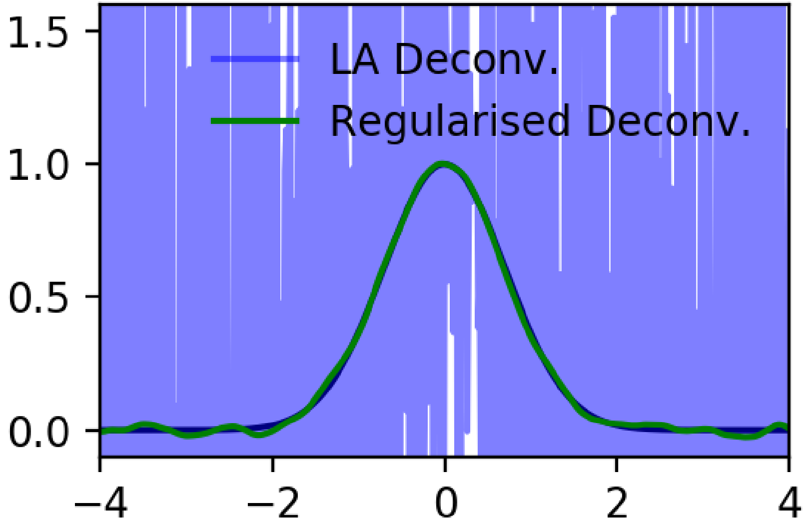
$$F(t) = \int R(\tau - t) f(\tau) d\tau \rightarrow \; F_i = R_{ij} f_j$$

- However, deconvolution is an ill-posed problem (R_{ij} is *band-limited*)
- Many regularised solution methods for deconvolution
- We will penalise large gradients:

$$\min_{\theta} \sum_i \left[y_i - R_{ij} f_j\right]^2 + \lambda \left(D^1_{ij} f_j\right)^2$$

First order finite difference





Non-Linear Least Squares

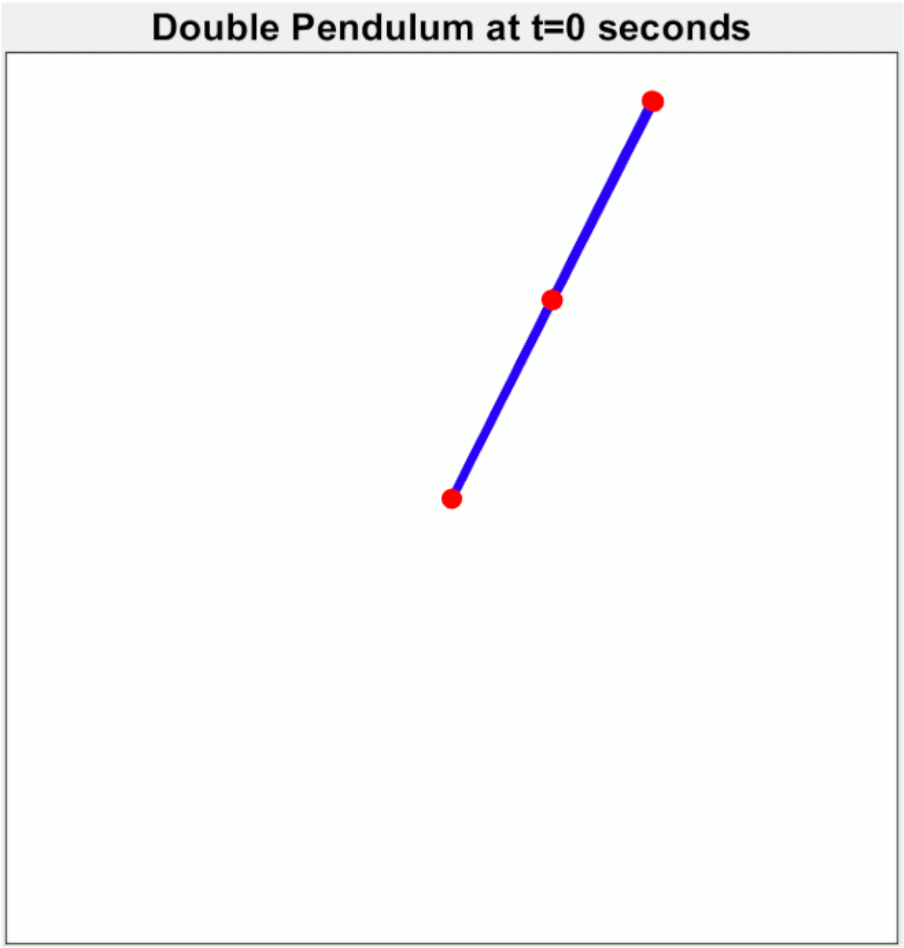
- Say we have a non-linear model with unknown parameters θ_j:

$$f\left(x, \theta_1 \ldots \theta_j \ldots \theta_N\right)$$

- And we are looking for the solution of least squares:

$$\min_{\theta} \sum_{i} w_i [y_i - f(x_i, \boldsymbol{\theta})]^2$$

- As you might expect, there is no direct linear algebra solution to this non-linear problem



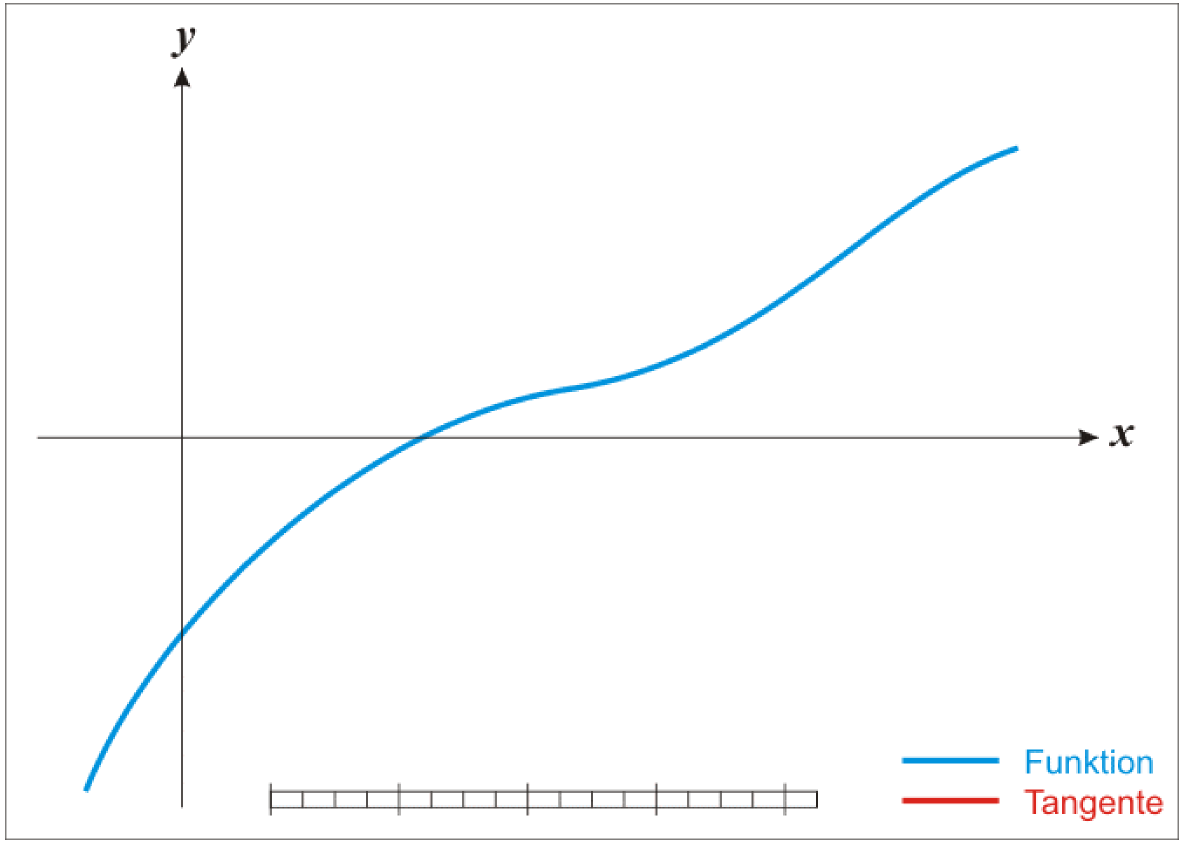
Reminder: Newton-Raphson Method

- Iterative method for finding the root of a function:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Minimising a function, g(x), is same as root finding its gradient function, f(x)

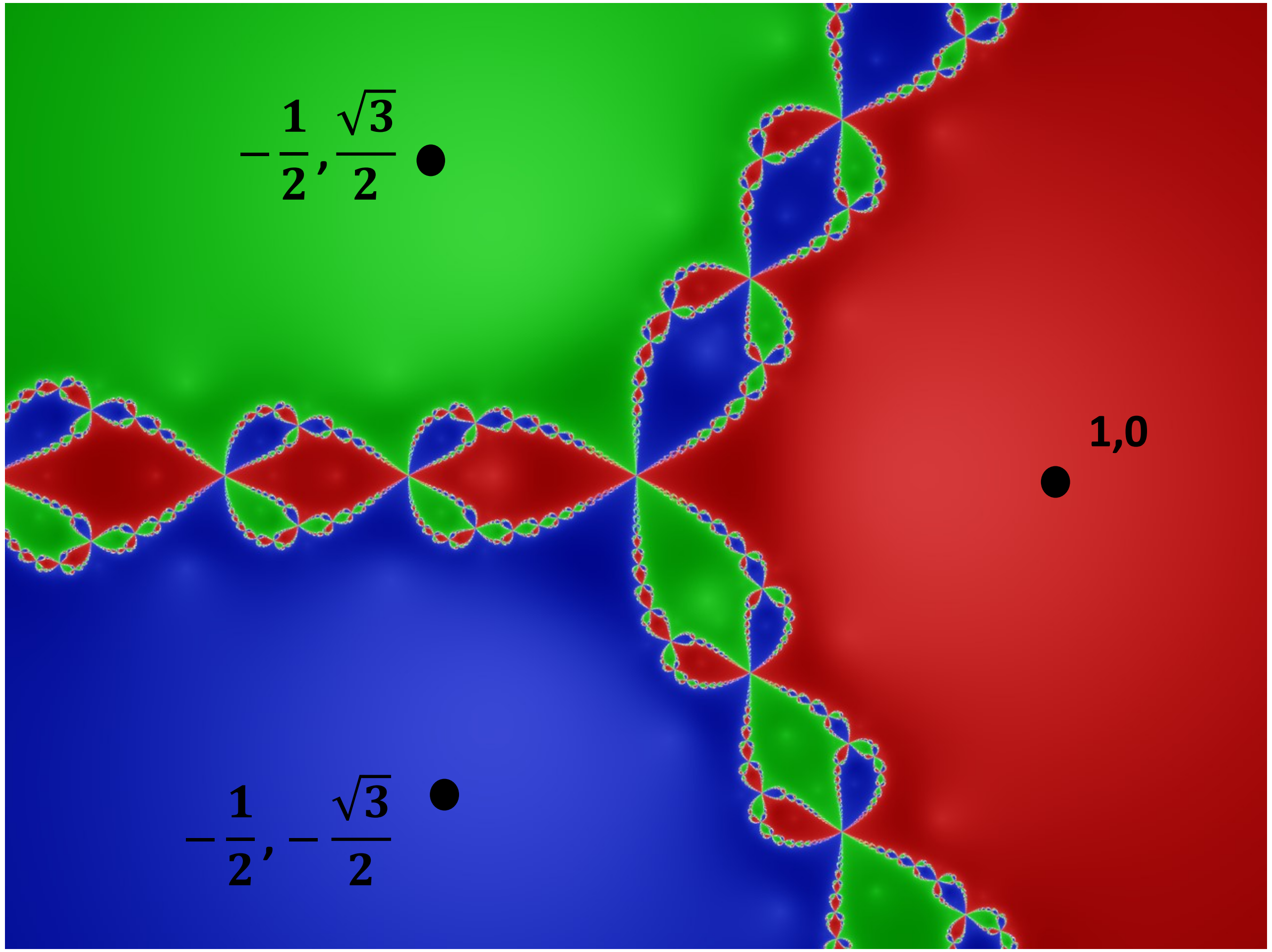
$$x_{n+1} = x_n - \frac{g'(x_n)}{g''(x_n)}$$



Fun aside: Newton's Fractal

- Depending on starting point, one converges to a different root
- Which root you end up at can have a fractal structure!
- Example: $f(z) = z^3 - 1$

- Example of Holomorphic Dynamics (cool name!)



Non-Linear Least Squares

- We are looking for the solution of least squares:

$$\min_{\theta} S(\boldsymbol{\theta}) = \min_{\theta} \sum_i w_i [y_i - f(x_i, \boldsymbol{\theta})]^2$$

- Lets linearise for small displacements in $\boldsymbol{\theta}$, and drop weights for simplicity:

$$\min_{d\theta} \sum_i [y_i - f(x_i, \boldsymbol{\theta} + \boldsymbol{d\theta})]^2$$

$$= \min_{d\theta} \sum_i \left[y_i - f(x_i, \boldsymbol{\theta}) + \underline{\boldsymbol{J}} \cdot \boldsymbol{d\theta}\right]^2$$

- Retrieving a multivariate version of Newton's method:

$$\rightarrow \boldsymbol{d\theta} = \left(\underline{\boldsymbol{J}}^T \underline{\boldsymbol{J}}\right)^{-1} \underline{\boldsymbol{J}}^T \left(\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\theta})\right)$$

N.B. we have approximated the Hessian as $2\underline{\boldsymbol{J}}^T\underline{\boldsymbol{J}}$, ignoring all 2nd derivatives, so we can get stuck in saddlepoints!

Non-Linear Least Squares

- Retrieving a multivariate version of Newton's method:

$$\rightarrow \boldsymbol{d\theta} = \left(\underline{\boldsymbol{J}}^T \underline{\boldsymbol{J}}\right)^{-1} \underline{\boldsymbol{J}}^T \left(\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\theta})\right)$$

- What if the Jacobian has large condition number? Regularize

$$\rightarrow \boldsymbol{d\theta} = \left(\underline{\boldsymbol{J}}^T \underline{\boldsymbol{J}} + \lambda \underline{\boldsymbol{I}}\right)^{-1} \underline{\boldsymbol{J}}^T \left(\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\theta})\right)$$

  - If $\underline{\boldsymbol{J}}^T \underline{\boldsymbol{J}}$ dominates, then Newton-Raphson step
  - If $\lambda \underline{\boldsymbol{I}}$ dominates, then gradient descent step

$$\rightarrow \boldsymbol{d\theta} = \frac{1}{\lambda} \cdot \underline{\boldsymbol{J}}^T \left(\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\theta})\right) = -\frac{1}{2\lambda} \frac{dS(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$$

N.B. Levenberg-Marquardt algorithm uses the above but adaptively changes $\boldsymbol{\lambda}$*, this is default algorithm in scipy.optimize.curve_fit*

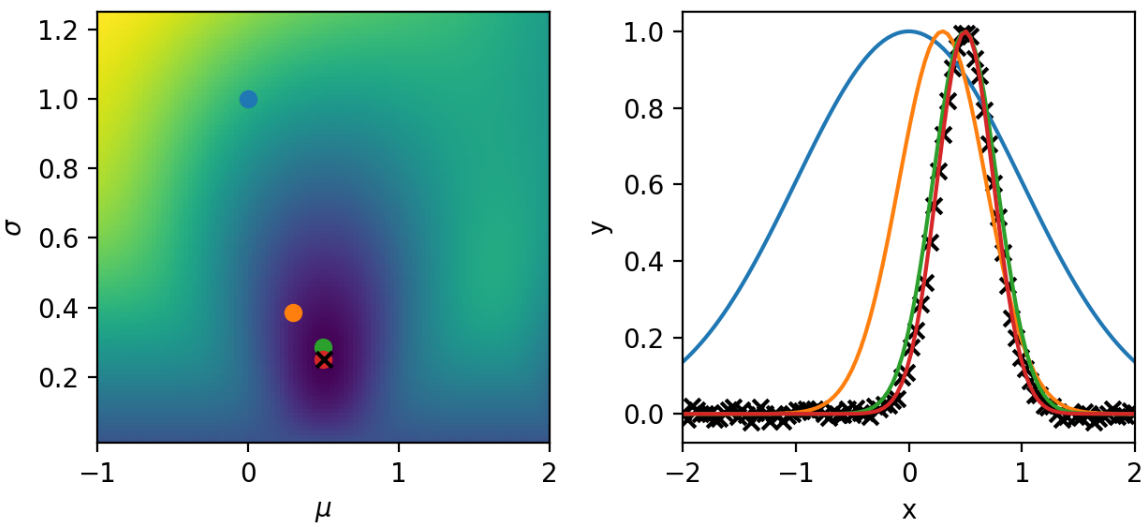
Non-Linear Least Squares

$$\rightarrow \boldsymbol{d\theta} = \left(\underline{\boldsymbol{J}}^T\underline{\boldsymbol{J}} + \lambda\underline{\boldsymbol{I}}\right)^{-1}\underline{\boldsymbol{J}}^T\left(\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\theta})\right)$$

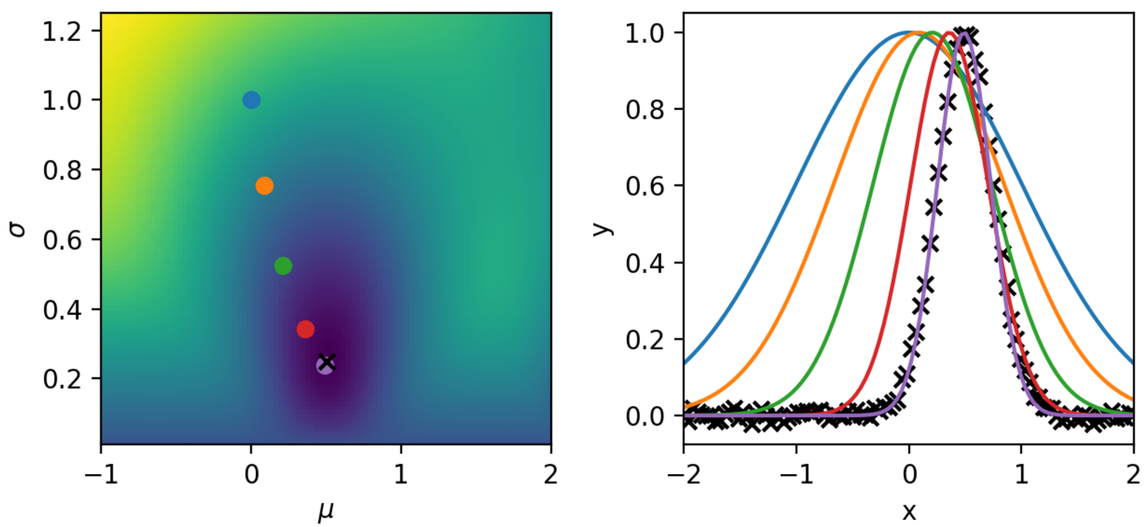
- If $\underline{\boldsymbol{J}}^T\underline{\boldsymbol{J}}$ dominates, then Newton-Raphson step
- If $\lambda\underline{\boldsymbol{I}}$ dominates, then gradient descent step

$$\rightarrow \boldsymbol{d\theta} = \frac{1}{\lambda}\cdot\underline{\boldsymbol{J}}^T\left(\boldsymbol{y} - f(\boldsymbol{x}, \boldsymbol{\theta})\right) = -\frac{1}{2\lambda}\frac{dS(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$$

Newton-Raphson



Gradient Descent



Differentiable Programming

- One might ask, how do you get the Jacobian, $\underline{\boldsymbol{J}}$, to perform gradient descent?

- Numerical derivatives: inaccurate and slow for large number of parameters

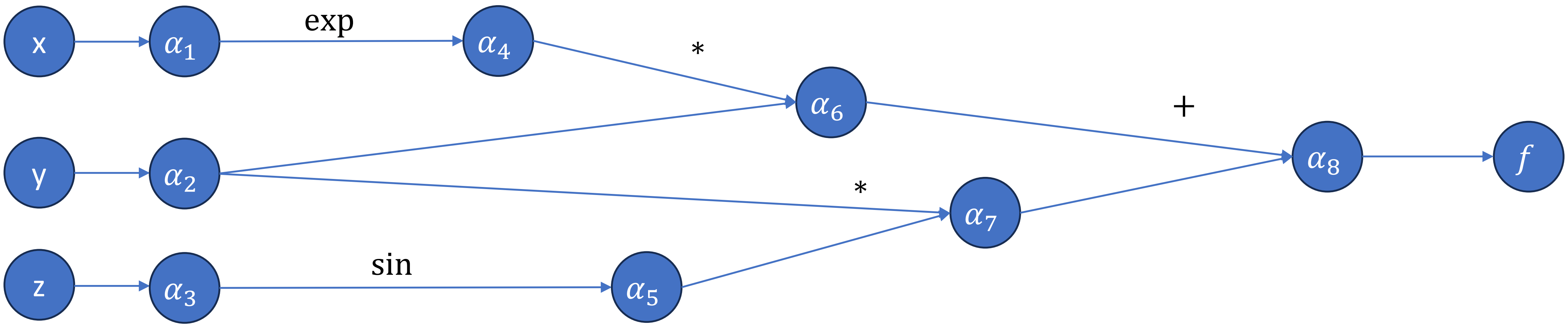
$$\frac{dy}{dx} \approx \frac{y(x+h) - y(x)}{h}$$

- Enter automatic differentiation

Automatic Differentiation Intro

- Every computer program can be written as a computation graph of mathematical 'primitive' functions (*+, x, exp, sin,* etc.)
- For example:

$$f(x,y,z) = y\,\exp(x) + y\,\sin(z)$$

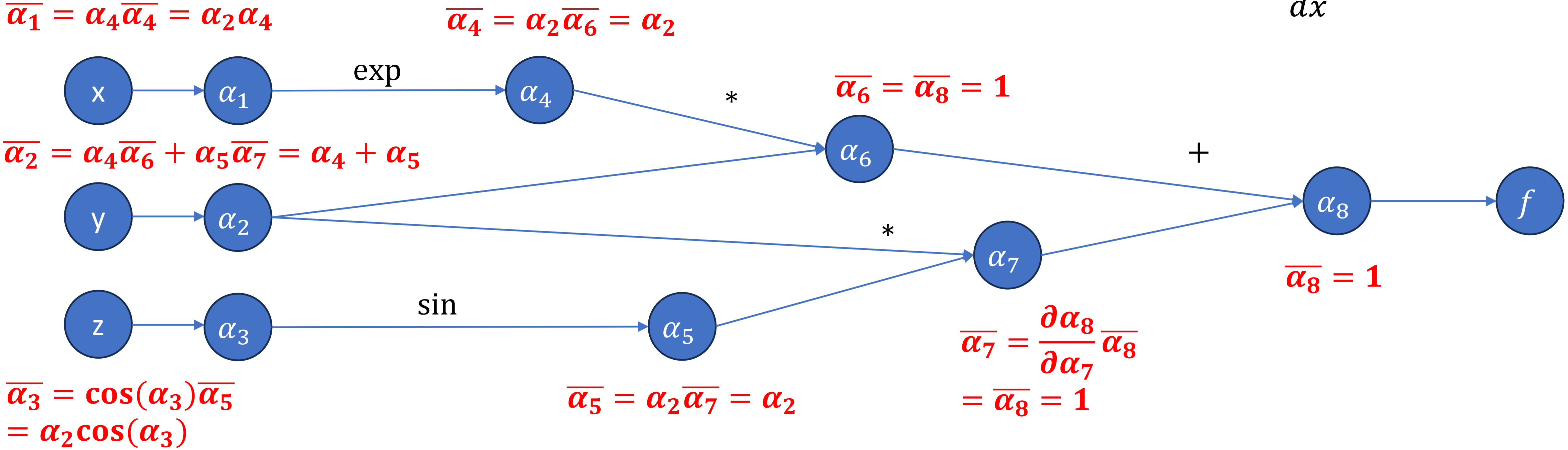


Automatic Differentiation Intro

- In our example:

Use the adjoint:

$$\bar{x} = \frac{df}{dx}$$



A single ‘back-propagation’ gives *all* gradients!

Using regression for inference

- Finding the ‘best fit’ is an optimisation problem

- In inference, we need additional information – the uncertainties

Bayes Theorem and Likelihoods

- Bayes' Theorem states:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{\text{Likelihood x Prior}}{\text{Evidence}}$$

- Canonical example, medical testing:

$$P(\text{ill}|+) = \frac{P(+|\text{ill})P(\text{ill})}{P(+)} = \frac{\text{Sensitivity x Prevalence}}{\text{True Positive} + \text{False Positive}}$$

- Observations always have uncertainty, therefore Bayes rule must be used in parameter estimation

Letter

LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S

Thomas Bayes

Published: **01 January 1763** | https://doi.org/10.1098/rstl.1763.0053

Abstract

Dear Sir, I Now send you an essay which I have found among the papers of our deceased friend Mr. Bayes, and which, in my opinion, has great merit, and well deserves to be preserved.

Bayesian Inference

- We wish to find the most likely physical parameters which describe the data:

P(model parameters given data): $P(\boldsymbol{\theta}|\boldsymbol{y}) \propto P(\boldsymbol{y}|\boldsymbol{\theta})P(\boldsymbol{\theta})$

- Define a likelihood that *given* the model, $f(\boldsymbol{\theta}, \boldsymbol{x})$, you would observe the data, say Gaussian errors:

$$P(\boldsymbol{\theta}|\boldsymbol{y}) \propto P(\boldsymbol{\theta}) \prod_i \exp\left(-\frac{\left(y_i - f(x_i, \boldsymbol{\theta})\right)^2}{2\sigma_i^2}\right)$$

- Take the logarithm to separate likelihood and priors

$$\propto \log(P(\boldsymbol{y}|\boldsymbol{\theta})) + \log(P(\boldsymbol{\theta})) = -\sum_i w_i[y_i - f(x_i, \boldsymbol{\theta})]^2 + R(\boldsymbol{\theta})$$

Least squares ***Regularisation***

Maximal posterior = Minimal least squares

Bayesian Inference – Laplace's Method

Pierre-Simon Laplace

- If we expand the log posterior about its optimum:

$$-\log\big(P(\boldsymbol{\theta}|\boldsymbol{y})\big) = F(\boldsymbol{\theta}) \approx F(\boldsymbol{\theta}^*) + \underbrace{J_F(\boldsymbol{\theta})}_{0}\,(\boldsymbol{\theta} - \boldsymbol{\theta}^*) + \frac{1}{2} H_F(\boldsymbol{\theta})\,(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^2$$

- We see the posterior is locally a multi-variate normal:

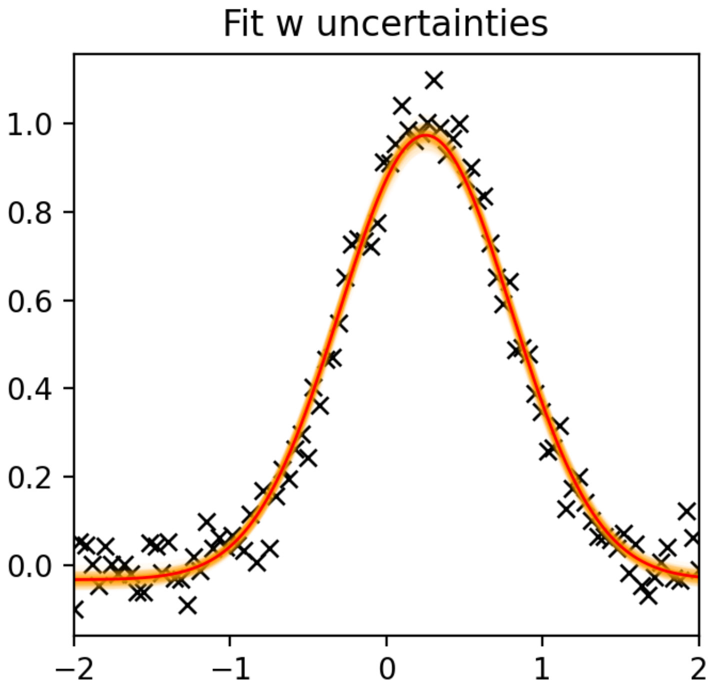
$$P(\boldsymbol{\theta}|\boldsymbol{y}) \sim \exp\left[-\frac{1}{2} H_F(\boldsymbol{\theta})\,(\boldsymbol{\theta} - \boldsymbol{\theta}^*)^2\right]$$

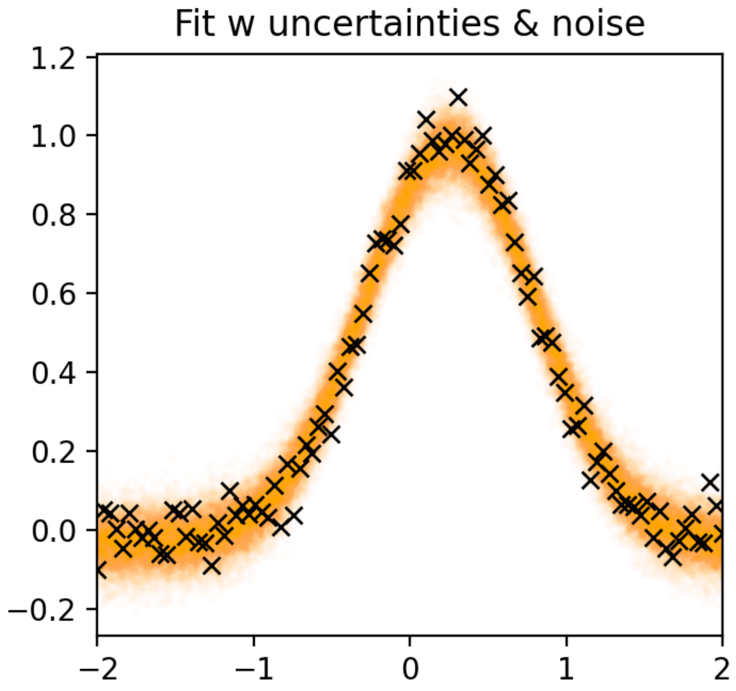
- The Hessian gives us the covariance matrix – this is what scipy's curve_fit is doing

Example in PythonJAX – Differentiable Programming

- Following our non-linear least squares example, lets fit a Gaussian to data with uncertainties estimated using Laplace's method

```
def neg_log_likelihood(mu,sig,A,B,x_data,y_data,yerr):
    """ Function to compute the negative log likelihood of our fit to data """
    y_model = Gaussian(x_data,mu,sig,A,B)
    return 0.5*jnp.sum(((y_data-y_model)/yerr)**2)

""" Gradient of neg_log_likelihood function w.r.t. input parameters """
# Here I am doing each gradient explicitly, but it can be done vectorially
# See how Hessian is done
gradL_mu  = jax.grad(neg_log_likelihood,argnums=0)
gradL_sig = jax.grad(neg_log_likelihood,argnums=1)
gradL_A   = jax.grad(neg_log_likelihood,argnums=2)
gradL_B   = jax.grad(neg_log_likelihood,argnums=3)

# N.B. we are going to perform gradient descent so
# the Jacobian of interest is the one with respect to the loss and not the model function
def Jacobian(mu,sig,A,B,x_data,y_data,yerr):
    """ Construct the Jacobian using the Automatic Differential gradient functions """
    J0 = gradL_mu(mu,sig,A,B,x_data,y_data,yerr)
    J1 = gradL_sig(mu,sig,A,B,x_data,y_data,yerr)
    J2 = gradL_A(mu,sig,A,B,x_data,y_data,yerr)
    J3 = gradL_B(mu,sig,A,B,x_data,y_data,yerr)
    J = jnp.array([J0,J1,J2,J3])
    return J

def Hessian(mu,sig,A,B,x_data,y_data,yerr):
    """ Construct the Hessian using the Automatic Differential gradient functions """
    H = jax.hessian(neg_log_likelihood,argnums=(0,1,2,3))(mu,sig,A,B,x_data,y_data,yerr)
    return jnp.array(H).reshape(4,4)

# Truth values of inputs
```



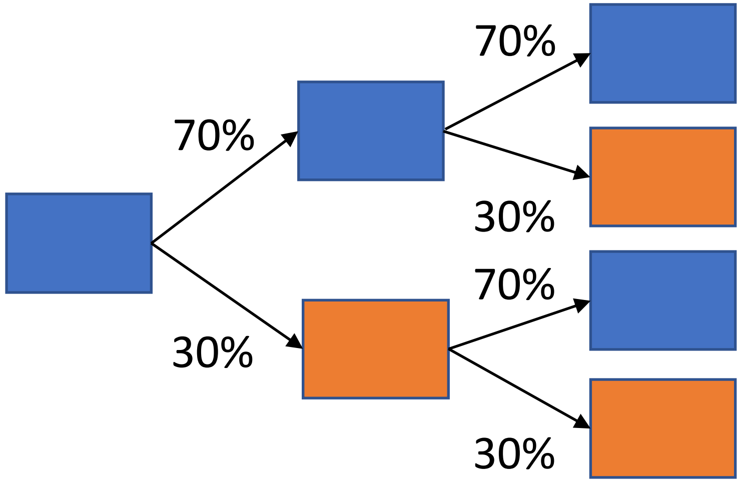


$$\underline{\Sigma} = \underline{H}^{-1}$$

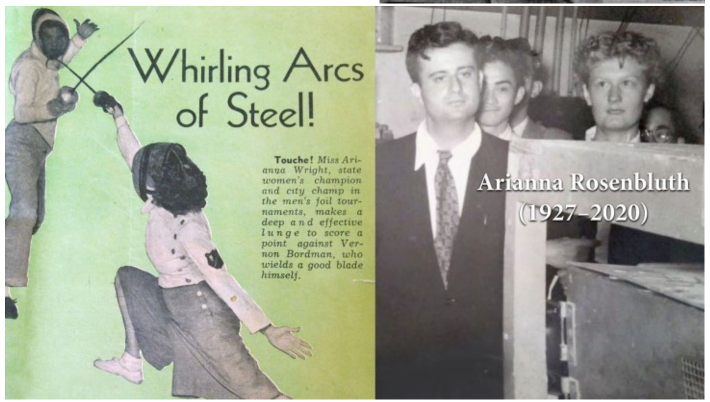
Bayesian Inference: Markov Chain Monte Carlo (MCMC)

- We wish to find the posterior distribution of some parameters but we can only evaluate a function *proportional* to it (Likelihood x Prior)

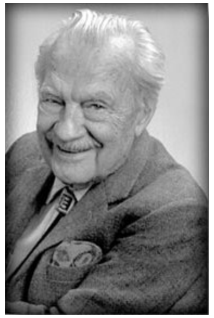
Markov chain:
A sequence of possible events where the next state only depends on the current state



Monte Carlo:
A class of numerical algorithm which involves repeated random sampling



- MCMC: Construct a Markov chain which has an equilibrium distribution of the posterior



Metropolis-Hastings* Algorithm

- $P(x)$ is the probability distribution function we want to sample, $f(x) \propto P(x)$:

1. Starting at x_i, randomly select a point x^* using jumping function $J(x^*|x_i)$
2. Compute the acceptance ratio, $\alpha = f(x^*)/f(x_i)$ and a random number $u \in [0,1]$
   a) If $u \leq \alpha$, accept $x^* \to x_{i+1}$
   b) Else, reject and $x_i \to x_{i+1}$

** Arguably, the Rosenbluth algorithm*

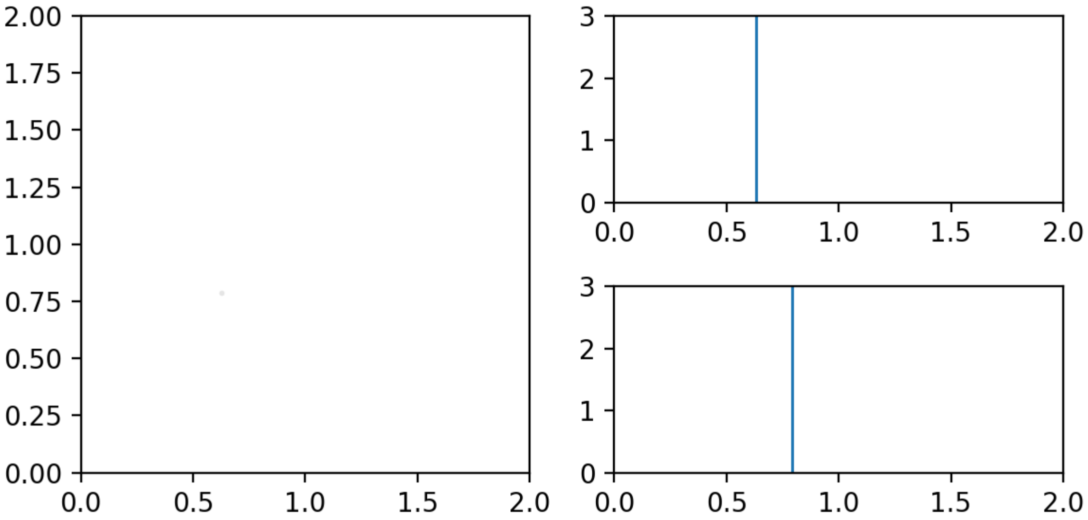
Equation of State Calculations by Fast Computing Machines

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, and Augusta H. Teller,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

and

Edward Teller,* *Department of Physics, University of Chicago, Chicago, Illinois*

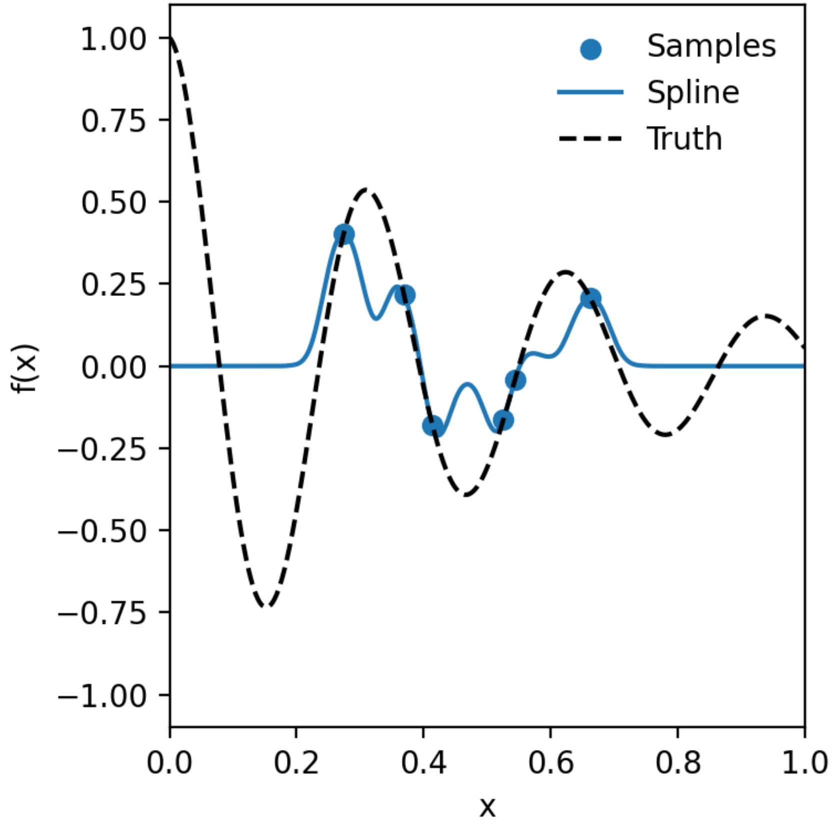
(Received March 6, 1953)




```
import numpy as np
import matplotlib.pyplot as plt

def f(x):
    d = x - mu
    arg = np.dot(d.T,np.dot(inv_corr,d))
    return np.exp(-0.5*arg)

def J(x):
    y = np.random.multivariate_normal(mean=x,cov=jump_size*np.eye(x.size))
    return y

# f parameters, multivariate Gaussian
mu   = np.array([1.0,1.0])
corr = np.array([[0.05,0.02],[0.02,0.05]])
inv_corr = np.linalg.inv(corr)

# Jump function properties
jump_size = 0.0025

# Metropolis algorithm with
chain_length = 50000
x = 2*np.random.rand(2)

x1_arr = np.array([])
x2_arr = np.array([])
u = 1.0
alpha = 0.0
for i in range(chain_length):
    while(u > alpha):
        y = J(x)
        u = np.random.rand()
        alpha = f(y)/f(x)
    x = y
    u = 1.0
    alpha = 0.0
    x1_arr = np.append(x1_arr,x[0])
    x2_arr = np.append(x2_arr,x[1])
```

Non-Parametric Regressors

- In the absence of a parametric model, one might want a generic fitting model
- Increasing in various levels of complexity:
  - Nearest-neighbour interpolation
  - Splines
  - Gaussian processes
- Fitting function modifies as you introduce more data? That's a nonparametric regressor



Spline fits are linear least squares:

$$f(x) = \sum_i a_i B(x_i, x)$$

Given data, construct B_{ij} at points x_j

Optimal amplitudes then:

$$\hat{\boldsymbol{a}} = \left(\underline{\boldsymbol{B}}^T \, \underline{\boldsymbol{B}}\right)^{-1} \underline{\boldsymbol{B}}^T \, \boldsymbol{y}$$

Conditioning multivariate normals

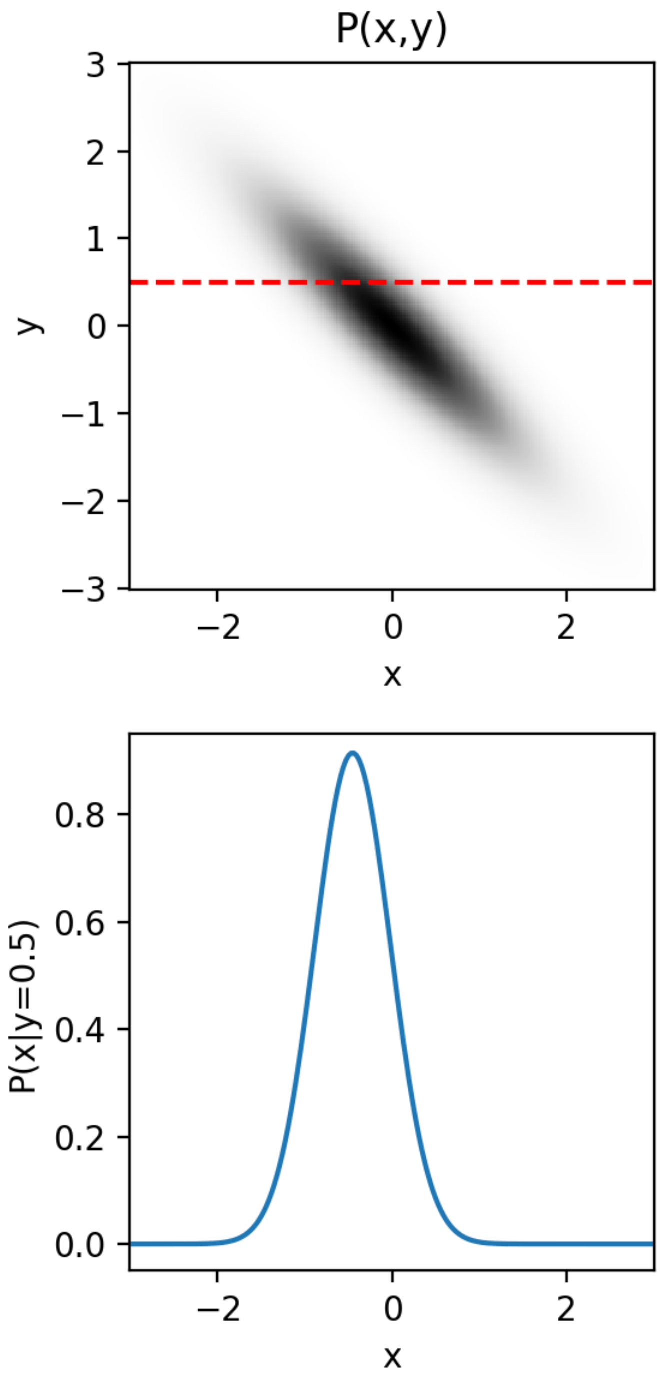
- We wish the predict the random variable Y_0 given data $\boldsymbol{Y}_n$, these will be correlated
- Assume joint distribution of prediction + data is multivariate normal:

$$\begin{pmatrix} Y_0 \\ \boldsymbol{Y}_n \end{pmatrix} \sim N_{1+n}\left[\begin{matrix} \mu_0 \\ \boldsymbol{\mu}_n \end{matrix}, \sigma^2 \begin{pmatrix} R_{11} & \boldsymbol{R}_{12} \\ \boldsymbol{R}_{21} & \boldsymbol{R}_{22} \end{pmatrix}\right]$$

- Conditional distribution of known data has mean and covariance matrix given by:

$$E(Y_0) = \mu_0 + \boldsymbol{R}_{12}\boldsymbol{R}_{22}^{-1}(\boldsymbol{Y}_n - \boldsymbol{\mu}_n)$$

$$\mathrm{Var}(Y_0) = \sigma^2(R_{11} - \boldsymbol{R}_{12}\boldsymbol{R}_{22}^{-1}\boldsymbol{R}_{21})$$



For more detail see: The Design and Analysis of Computer Experiments by Santner, Williams and Notz

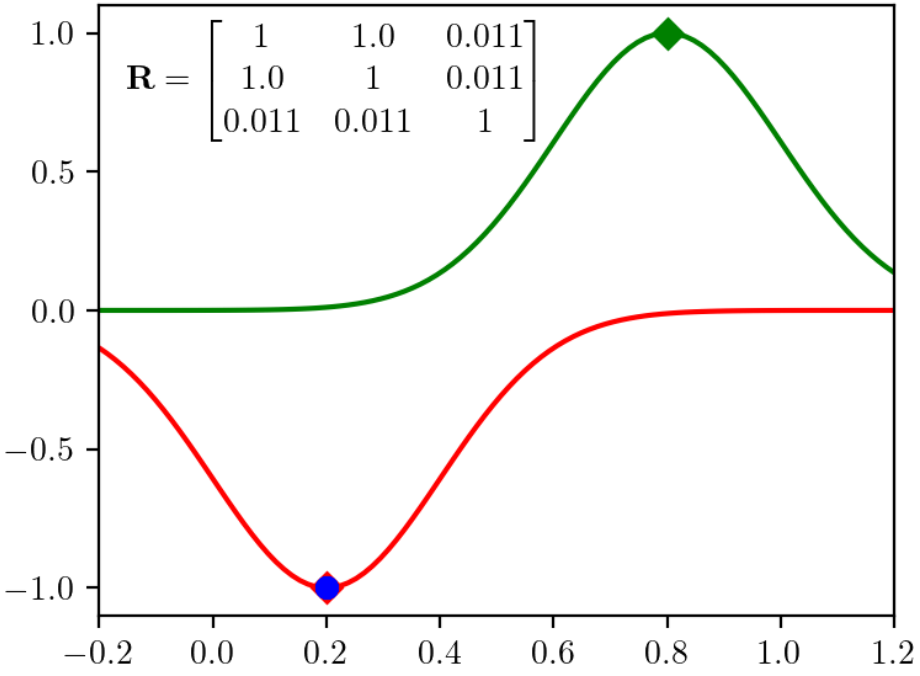
Gaussian Processes

$$\underline{\boldsymbol{\Sigma}} = \sigma^2 \begin{bmatrix} R(x_0 - x_0) & R(x_1 - x_0) & \cdots & R(x_N - x_0) \\ R(x_0 - x_1) & R(x_1 - x_1) & \cdots & R(x_N - x_1) \\ \vdots & \vdots & \ddots & \vdots \\ R(x_0 - x_N) & R(x_1 - x_N) & \cdots & R(x_N - x_N) \end{bmatrix}$$

- Gaussian processes use the conditional distribution to predict unobserved points
- Mean function (usually 0) is linear fitting function, $\mu(x) = \sum_i \theta_i f_i(x)$
- Correlation matrix defined by ‘kernel’ function $R(d)$

$$\boldsymbol{Y}(x) = \boldsymbol{\mu}(x) + \boldsymbol{Z}(\mu = 0, R(d))$$

- Correlation function quantifies how correlated points in space are
- How do you pick kernel free parameters…



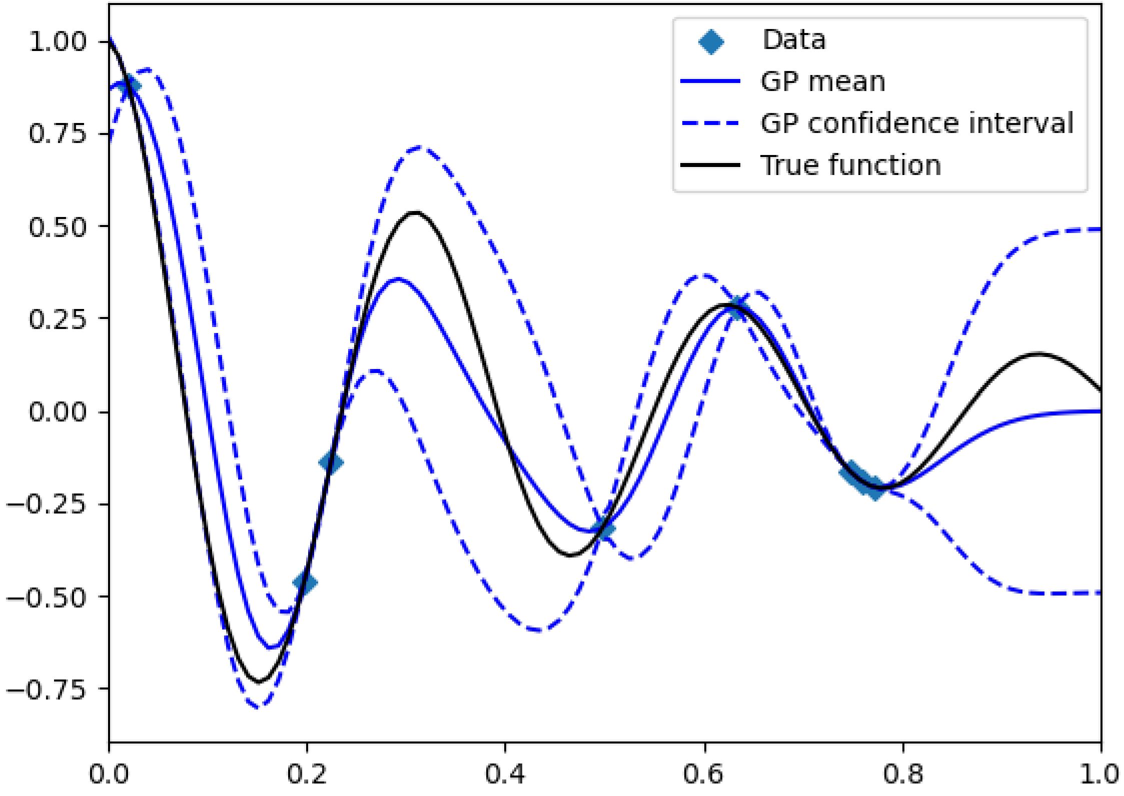
$$E(Y_0) = \boldsymbol{R}_{12}\boldsymbol{R}_{22}^{-1}\boldsymbol{Y}_n$$

$$R(d, l) = \exp\left[-\frac{1}{2}\left(\frac{d}{l}\right)^2\right]$$

Gaussian Processes: Example

```python
def correlation_function(x1,x2,theta):
    """ Correlation function or kernal of the GP, we will use a Gaussian with length scale theta """
    return np.exp(-0.5*((x1-x2)/theta)**2)

def correlation_matrix(x,theta):
    """ Construct correlation matrix using correlation function """
    x1,x2 = np.meshgrid(x,x)
    corr = correlation_function(x1,x2,theta)
    return corr

def sigma2_MLE(corr,y):
    """ Maximum Likelihood estimation of vertial length scale sigma
        see pg 65-66 of Santner, Williams and Notz"""
    n = y.shape[0]
    return np.dot(y,np.matmul(np.linalg.inv(corr),y))/n

def neg_log_likelihood(theta,x,y):
    """ Negative Log Likelihood of Gaussian process for a given choice of length scale theta
        see pg 65-66 of Santner, Williams and Notz"""
    n = x.shape[0]
    corr = correlation_matrix(x,theta)
    sigma2 = sigma2_MLE(corr,y)
    log_l = n*np.log(sigma2)+np.log(np.linalg.det(corr))
    return log_l

def conditional_distribution(x_trial,x_known,y_known,sigma,theta):
    """ Calculate p(x_trial | x_known,y_known) for given values of sigma and theta """
    x_total = np.concatenate((x_trial,x_known),axis=0)
    N_trial = x_trial.shape[0]
    corr    = sigma**2*correlation_matrix(x_total,theta)
    corr_11 = corr[:N_trial,:N_trial]
    corr_12 = corr[:N_trial,N_trial:]
    corr_21 = corr_12.T
    corr_22 = corr[N_trial:,N_trial:]
    # Note unconditional mu = 0
    corr_22_inv = np.linalg.inv(corr_22)
    mu_cond  = np.dot(np.matmul(corr_12,corr_22_inv),y_known)
    sig_cond = corr_11 - np.matmul(corr_12,np.matmul(corr_22_inv,corr_21))
    return mu_cond,sig_cond

res = minimize(neg_log_likelihood,0.1,args=(X_known,Y_known))

theta_opt = res.x[0]
corr_opt  = correlation_matrix(X_known,theta_opt)
sigma_opt = np.sqrt(sigma2_MLE(corr_opt,Y_known))

X_trial = np.linspace(0.0,1.0,100)

mu_cond,sig_cond = conditional_distribution(X_trial,X_known,Y_known,sigma_opt,theta_opt)
```

Kernel parameters found by maximal likelihood estimation given known data
Python libraries: *Gpy, sklearn*



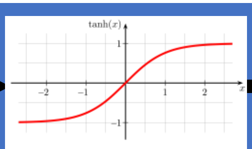
Neural Networks

- Neural networks are the canonical AI model
- They are black-box functions with many, many learnable parameters, *ChatGPT 175 billion*
- Many flavours, we will discuss the multi-layer perceptron (MLP)
- Basic unit is the neuron:

Input: x

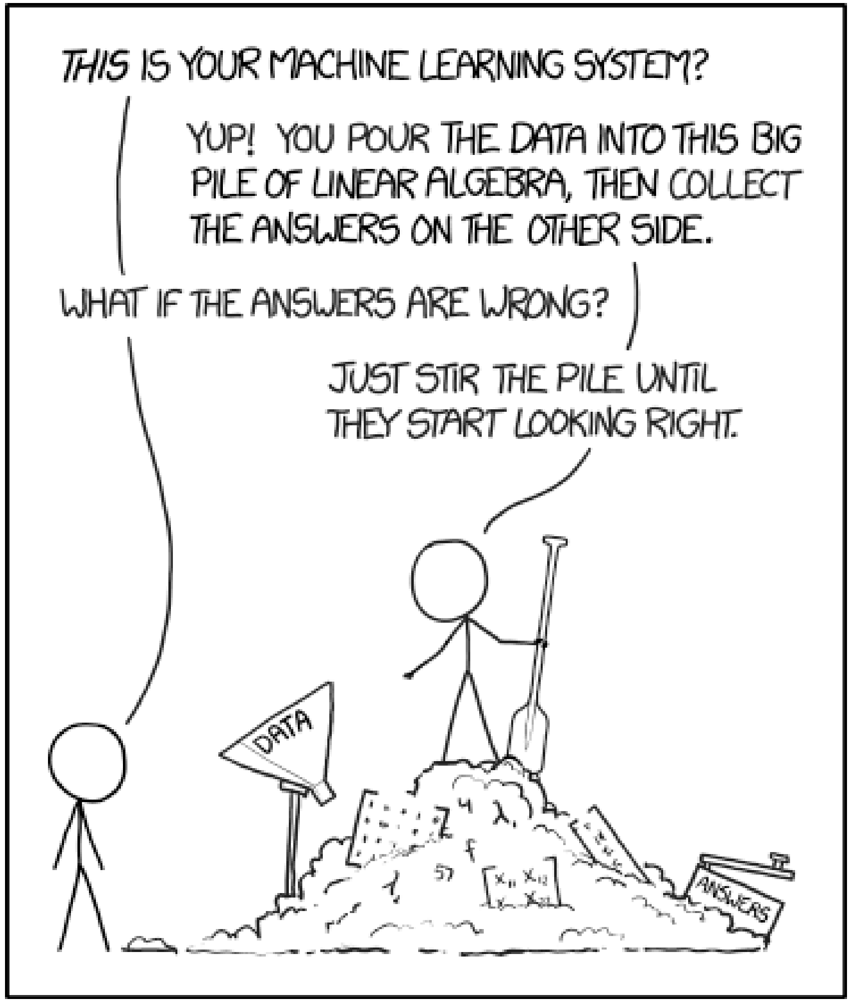
Neuron properties:
Parameters: weight w**, bias** b
Activation function: tanh, ReLU, etc.

$$x' = wx + b$$

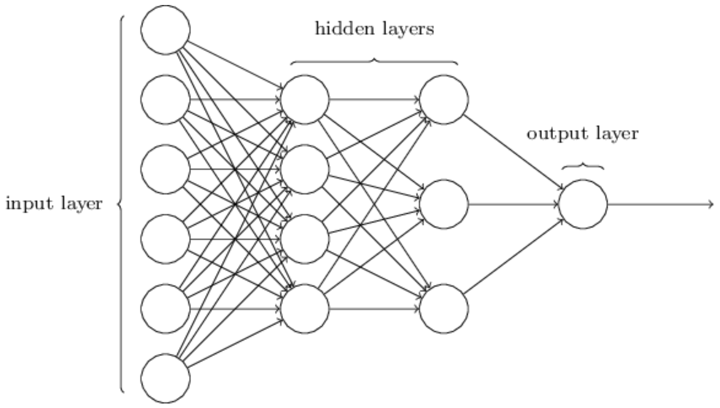


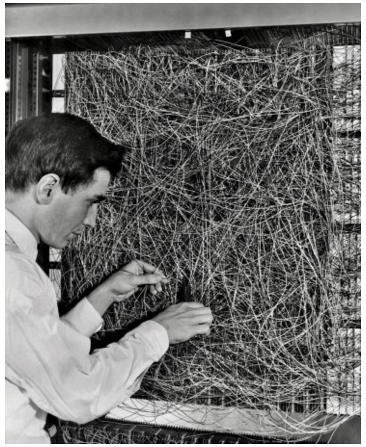
y

Output: y



Multi-Layer Perceptron





Rosenblatt with the Mark 1 Perceptron

- Stack neurons into layers and *fully*-connect layers
- Define loss function between predicted and real outputs e.g. mean-squared-error (MSE)
- Update neuron parameters by gradient descent (or variations of it)
- It is just regularised non-linear least squares again!

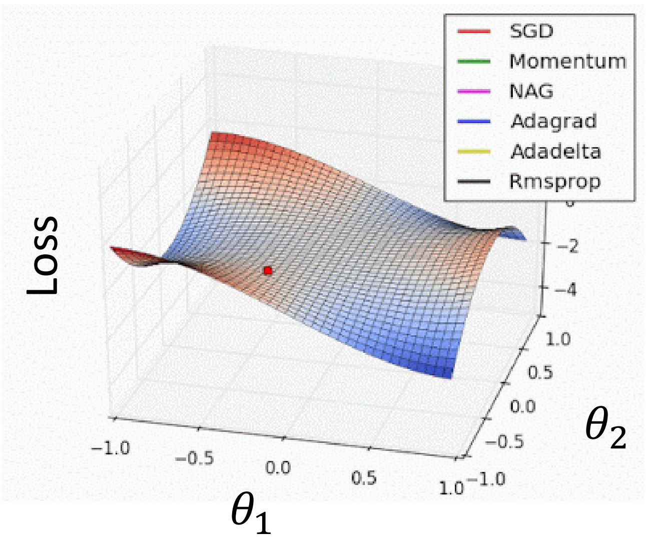
$$\boldsymbol{d\theta} = \alpha \cdot \underline{\boldsymbol{J}}^T\left(\boldsymbol{y} - f\left(\boldsymbol{x}, \boldsymbol{\theta}\right)\right)$$

Learning rate

Jacobian found by AD

Neural network

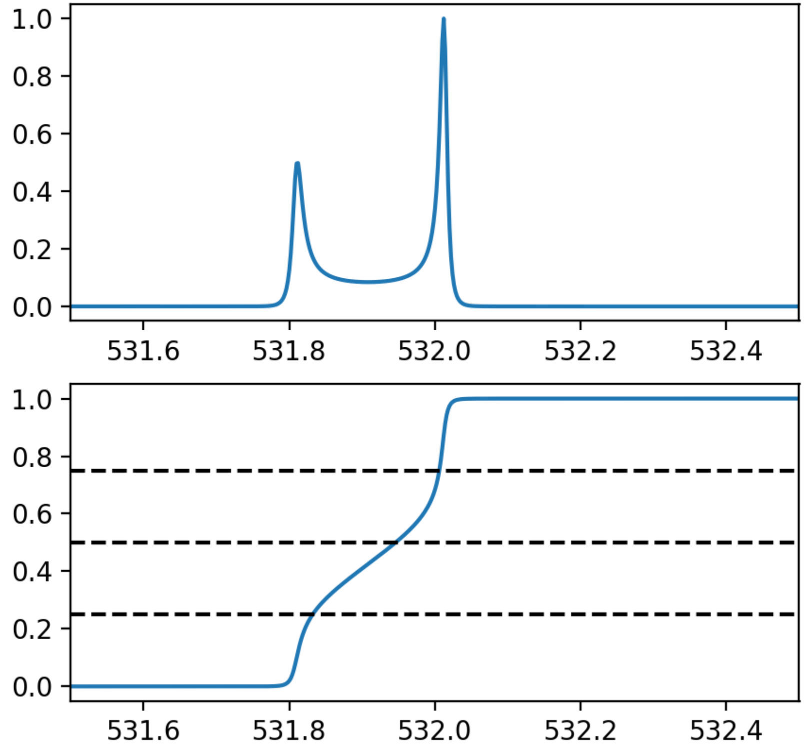
Various optimisers



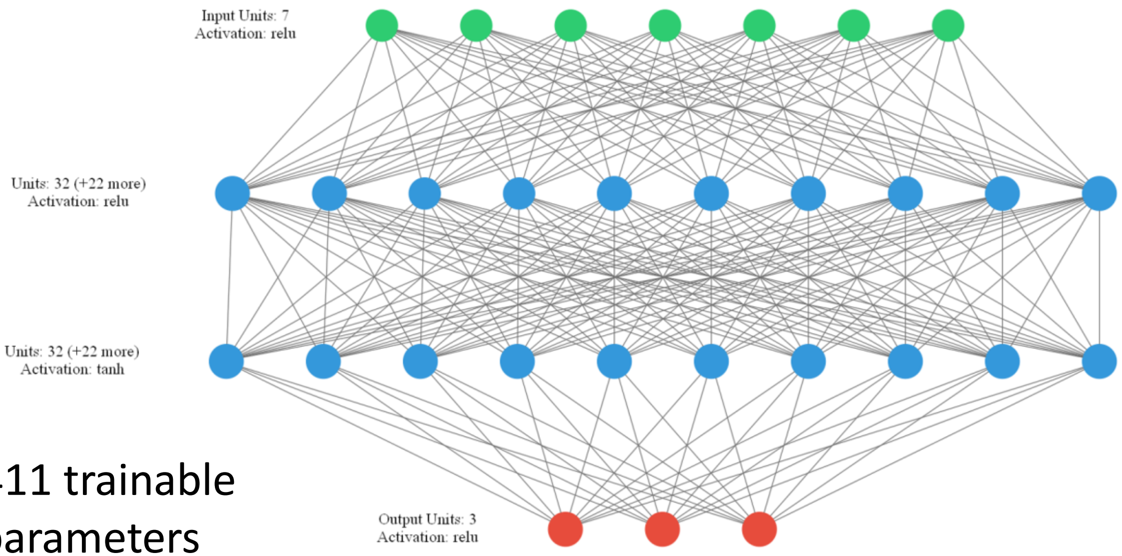
Neural Networks: Example

- Learn relationship between Si IAW Thomson scattering features and physical parameters

$$\boldsymbol{\theta} = (n_e, \mu, T_e, T_i, Z, v_{fi}, v_{fe})$$
$$\boldsymbol{y} = (\Delta\lambda_m, \Delta\lambda_h - \Delta\lambda_l, a_\lambda)$$



- Selection of the NN architecture can be human judgement or algorithmic
  - Tweaking of model parameters becomes tweaking of hyper-parameters
- Data ‘featurisation’ is often key in producing a good model



1,411 trainable parameters

Neural Networks: Example

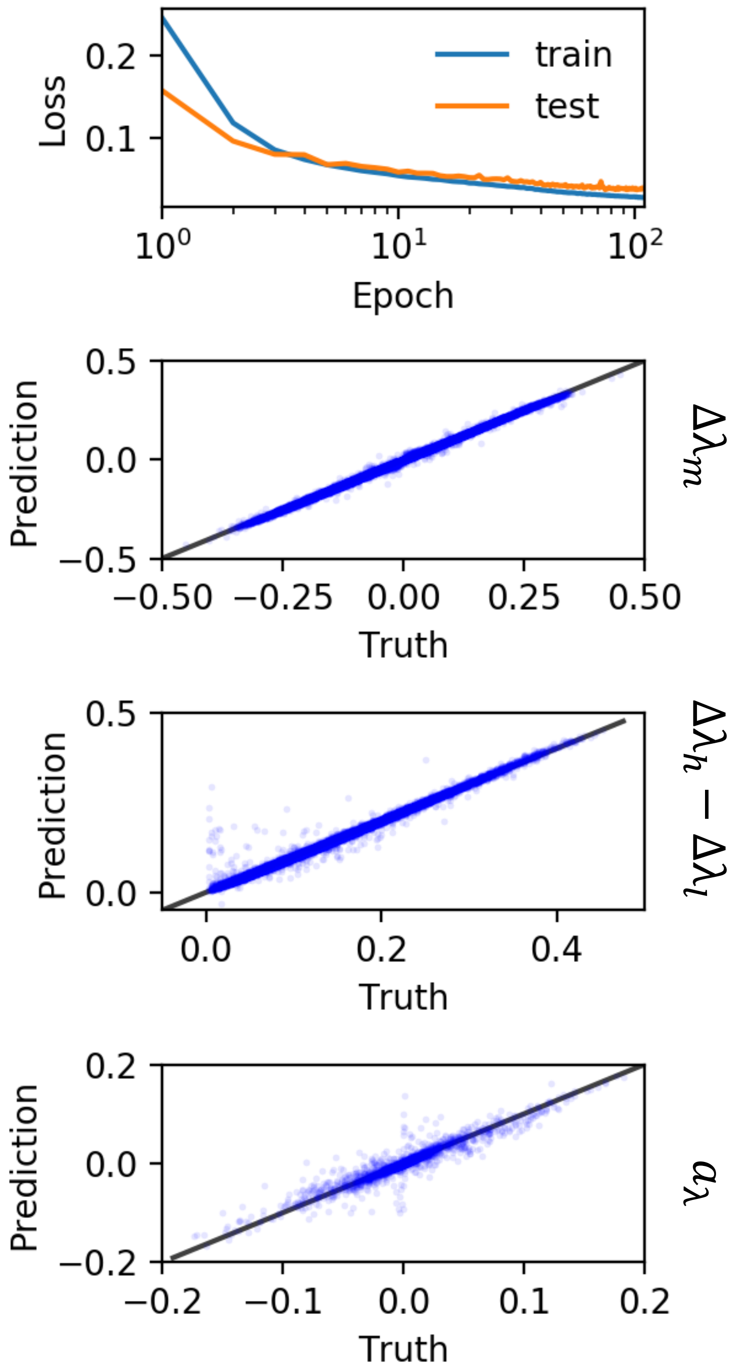
- Learn relationship between Si IAW Thomson scattering features and physical parameters

$$\boldsymbol{\theta} = (n_e, \mu, T_e, T_i, Z, v_{fi}, v_{fe})$$
$$\boldsymbol{y} = (\Delta\lambda_m, \Delta\lambda_h - \Delta\lambda_l, a_\lambda)$$

- Deep learning libraries: *keras/tensorflow, sklearn, pytorch*

```
# define the keras model
model = Sequential()
model.add(InputLayer(input_shape=(input_size,)))
model.add(Dense(32, input_shape=(input_size,), activation='relu'))
model.add(Dense(32, activation='tanh'))
model.add(Dense(output_size))

model.summary()

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

https://gb.coursera.org/learn/neural-networks-deep-learning



Examples in the field

Transfer learning in ICF experiments

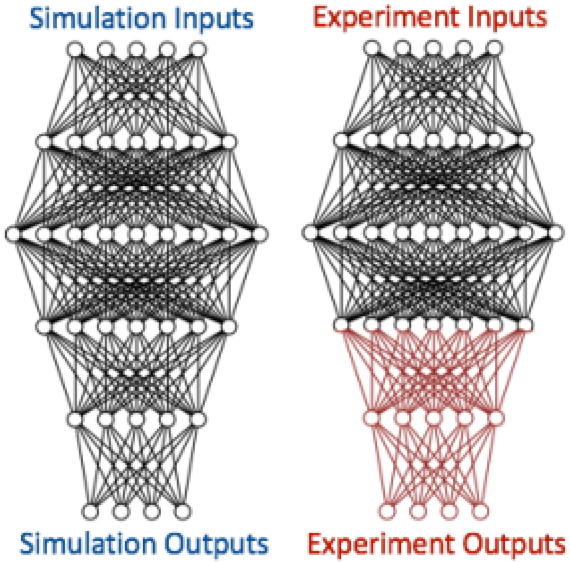
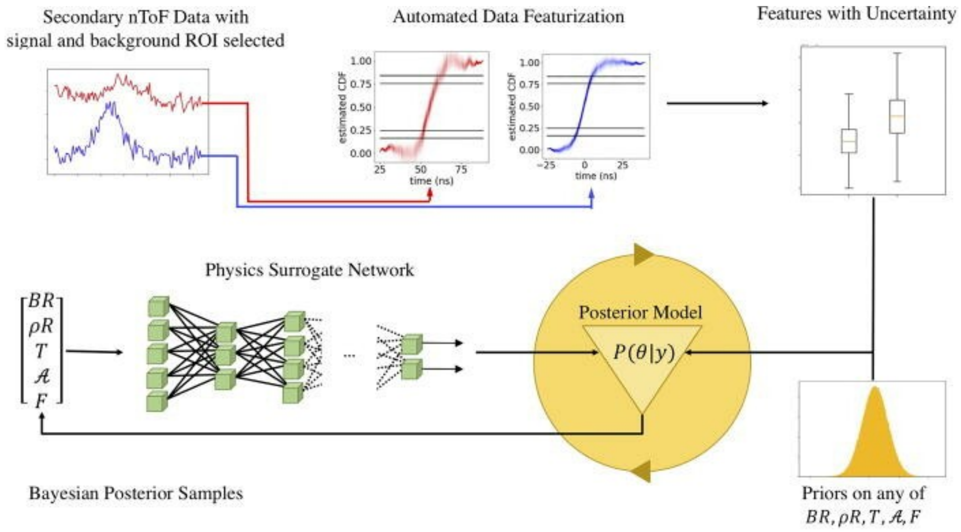
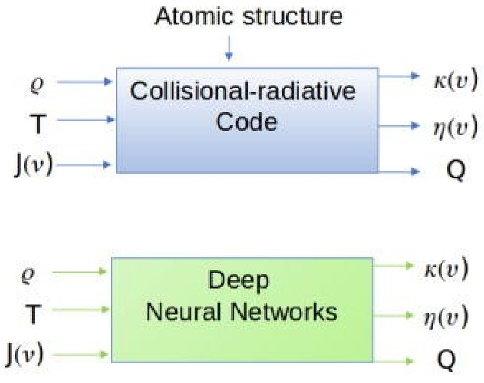


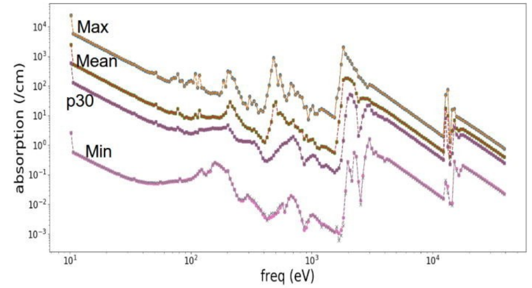
Fig. 1. To transfer learn from simulations to experiments, the first three layers of the simulation-based network are frozen, and the remaining two layers are available for retraining with the experimental data.

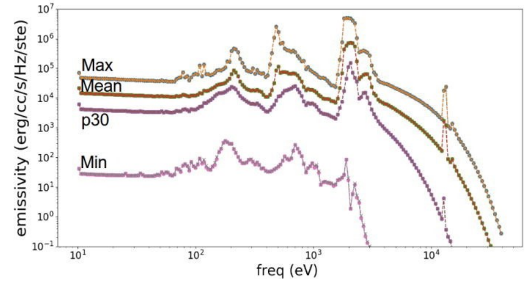
Surrogate model of secondaries in MagLIF



NLTE opacity emulators







Method/topic	Resource (Youtube, book, python library, course, etc.)
Machine Learning	https://github.com/josephmisiti/awesome-machine-learning - Curated list of ML resources SciKit-Learn (sklearn), pycaret package gives examples Scipy.optimize “Pattern Recognition and Machine Learning” book
Bayesian Statistics	“Statistical Rethinking” Youtube lecture series and book
Markov Chain Monte Carlo	PyMC, emcee
Gaussian Processes	GPy, GPyTorch, sklearn “Design and Analysis of Computer Experiments” book (WARNING: Mathsy)
Deep Learning	**Andrew Ng’s Deep Learning specialisation** on Coursera (Can be done for free within time window, otherwise monthly charge)
Neural Networks	Tensorflow, PyTorch, Keras
Differentiable Programming	JAX and associated libraries (diffrax, optax, …)
Optimisation	Convex Optimisation I and II, Stanford Engineering Everywhere
Software Engineering	Free O’Reilly access: https://www.oreilly.com/library-access/ - *Python Cookbook* Imperial’s ‘Essential Software Engineering for Researchers’ course (1 credit)

Technique	Pros	Cons
Traditional optimisation (e.g. least-squares with L-M)	Widely used and easy to start Low number of hyperparameters Familiar techniques	Cannot handle noisy data Often require known Jacobian for best performance Uncertainties use Laplace's method Results depend on starting location Scales poorly Can require regularisation
Gradient free optimisation (e.g. Bayesian optimisation)	Finds global minimum No need for gradient information Can handle noisy data	Scales very poorly Choice of kernel (BO) Spend time exploring domain
Differentiable programming	Gradient information for any program Scales well Allows gradients to appear in loss function	Must write code in differentiable form – harder to debug
Splines, etc.	Flexible functions with local extent Data driven predictor	Often want model parameters not predictor function
Markov Chain Monte Carlo	Explore full posterior distribution Proper Bayesian treatment Does not require gradients (apart from HMCMC)	Large number of function evaluations needed
Gaussian processes	Like splines but with uncertainties	Scales poorly
Neural networks	Universal function approximators Scales very well Architectures adapted to problems e.g graphs for molecules	Requires large data set Large number of hyperparameters Model training can be expensive

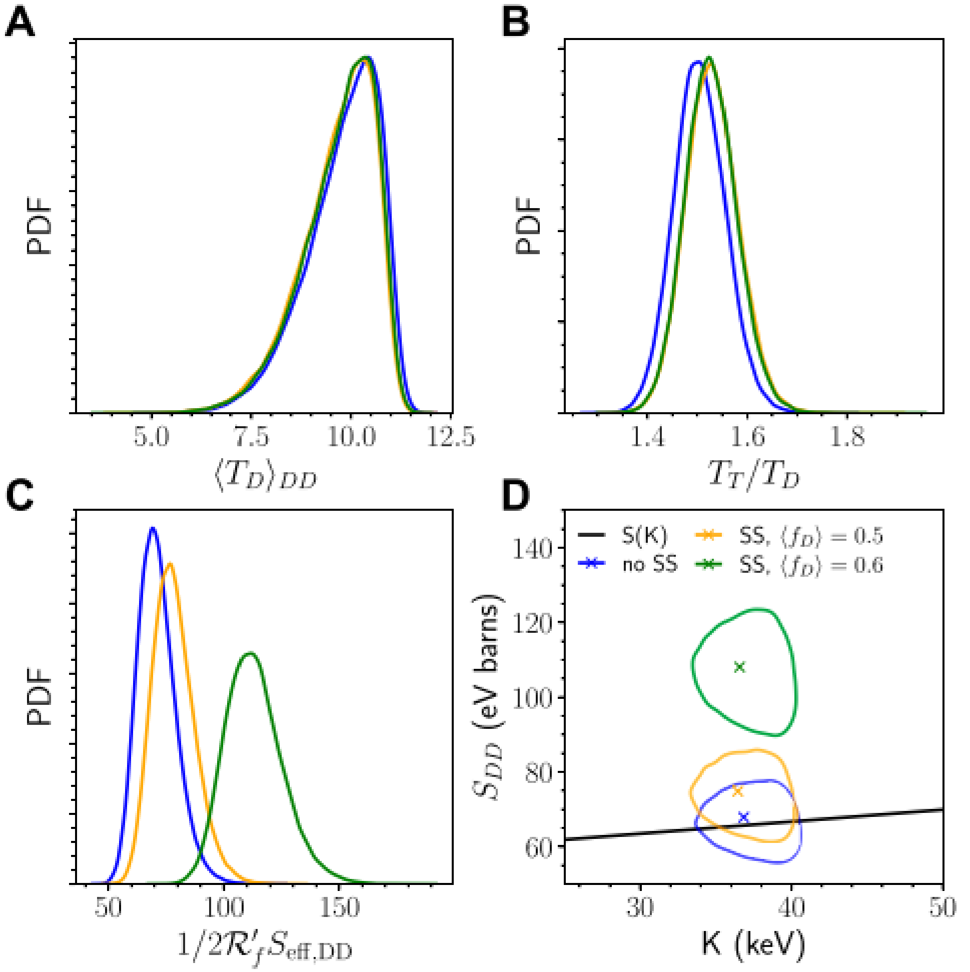
Machine Learning in Plasma Group Research

- Four examples of Plasma Group publications using ML:
  - “Efficacy of inertial confinement fusion experiments in light ion fusion cross section measurement at nucleosynthesis relevant energies”
  - “Automation and control of laser wakefield accelerators using Bayesian optimization”
  - “Laser Wakefield Accelerator modelling with Variational Neural Networks”
  - “Monte Carlo modelling of the linear Breit-Wheeler process within the Geant4 framework”
- These cover Markov Chain Monte Carlo, Gaussian Processes, Neural Networks and optimisation techniques

Efficacy of inertial confinement fusion experiments in light ion fusion cross section measurement at nucleosynthesis relevant energies (Crilly et al Frontiers in Physics 2022)

This research looked to answer the question, given experimental uncertainties and model assumptions, how well can we constrain fusion cross sections using ICF experiments:

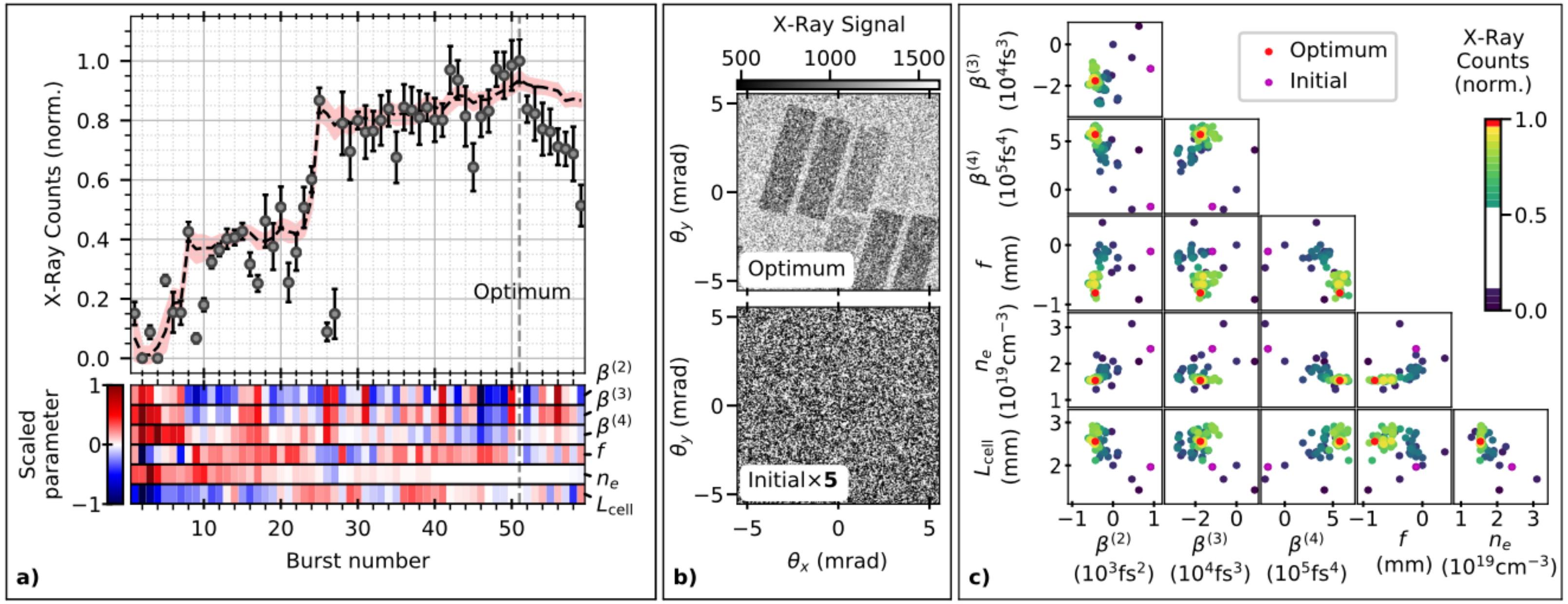
1. Produce synthetic data using full model
2. Including typical experimental uncertainties, fit synthetic data using reduced model
3. Explore the posterior distribution using MCMC
4. Demonstrate degeneracies and biases in reduced models



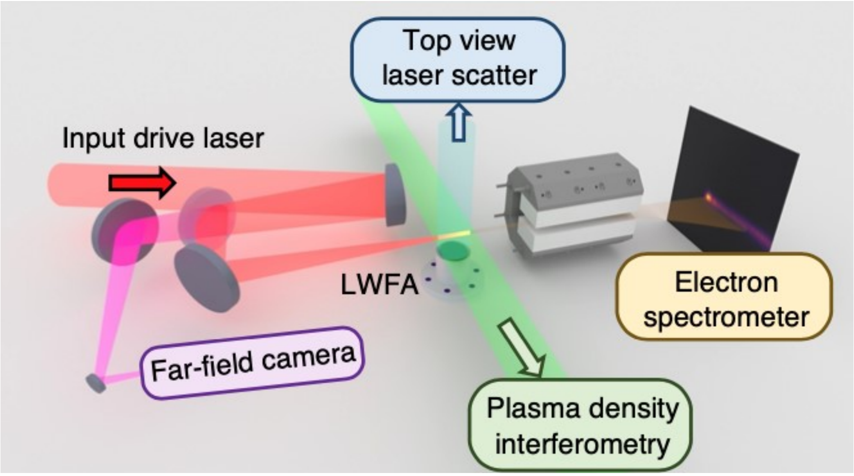
Automation and control of laser wakefield accelerators using Bayesian optimization (Shalloo et al Nat. Comms 2020)

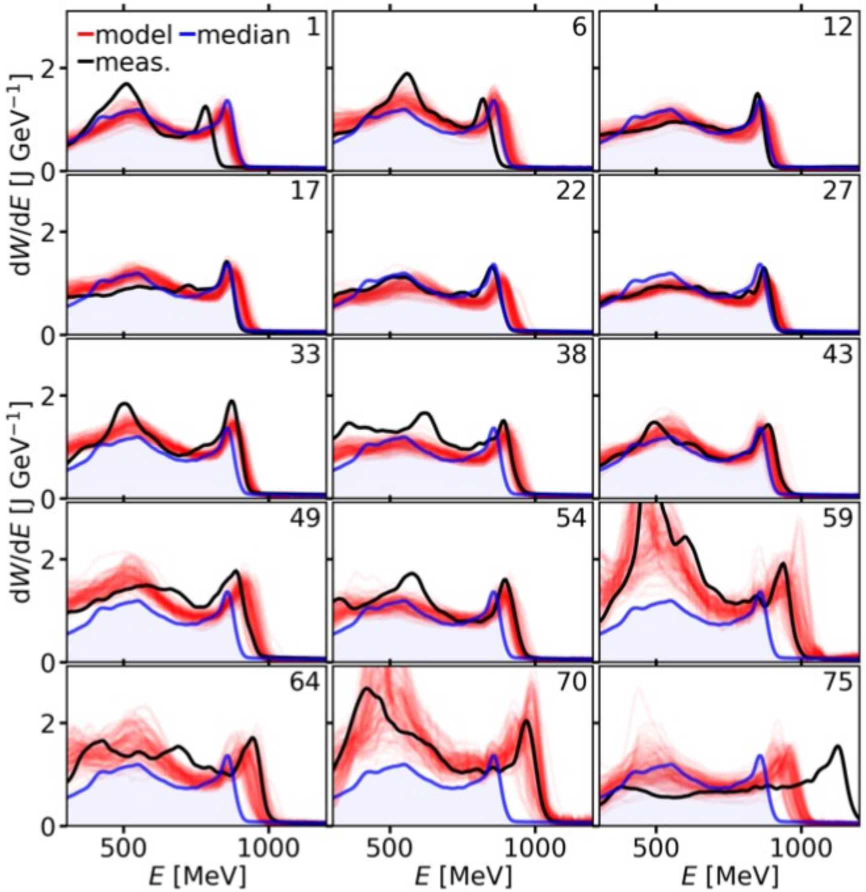
1. Uses Gaussian Process Regression to optimize some property of a laser wakefield accelerator
2. Experimental measurements are made at initial positions
3. GPR model updated with the measurements to form a posterior distribution.
4. An acquisition function is computed and used to select the next measurement location.
5. Steps 3–4 are repeated until the convergence criteria are met.
6. Acquisition function chosen to balance finding the optimum and exploring the function

4 laser parameters and 2 plasma parameters controlled to optimize the brightness of generated X-rays



Laser Wakefield Accelerator modelling with Variational Neural Networks (Streeter et al HEDP 2023)





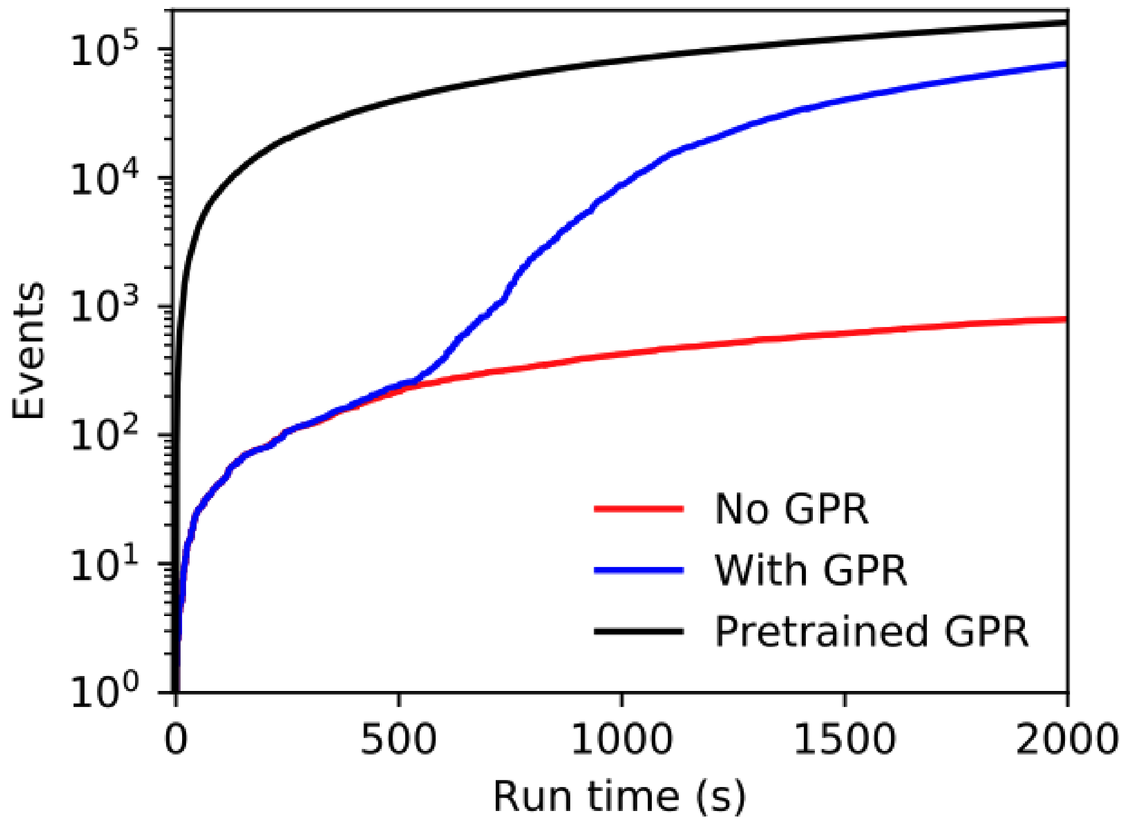
- LWFA spectra differ from shot-to-shot
- In experiments that use the beam we would like to know the spectrum on each shot
    - But if the beam is used, can we use other diagnostics to work out what the beam was?
- We trained a variational neural network based on three non-invasive diagnostics to see if this was feasible
    - Variational neural networks free parameters are random normal variables
    - E.g. Weight $w \rightarrow (\mu_w, \sigma_w)$

15 example predictions – compared to the actual measurement
Black: measured spectrum; Red: model prediction;
Blue: previous best estimate without using a NN

Monte Carlo modelling of the linear Breit-Wheeler process within the Geant4 framework

R Watt https://arxiv.org/pdf/2302.04950.pdf

- Calculation of number of e+ e- pairs produced when high energy gamma rays pass through laser produced x-ray field
- Calculation involves a triple integral – computationally expensive
  - Common solution would be to use look up table, but this wasn't suitable
  - Instead, we use full calculations to train a GPR model "on-the-fly".
  - If error in GPR function is too high for a point in parameter space do full calculation
  - If its already low enough use the GPR model
- This produces a significant speed up in the calculation after initial "training" phase



Links

Code and slides
URL:
https://github.com/aidancrilly
/ML_Lecture_Demos