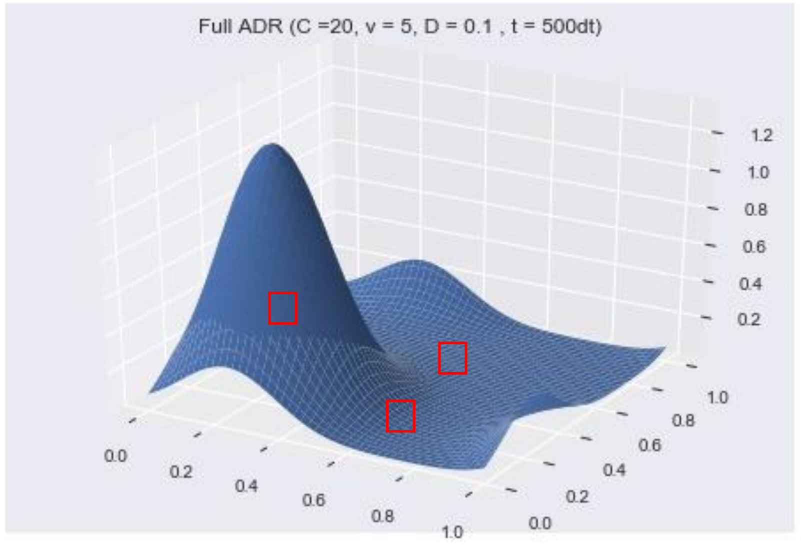
Machine Learning for Partial Differential Equations

-Shaan Desai

Introduction

- Focus so far: going from PDE’s to numerical solutions (e.g. Burgers, ADR, SE, SDE..)
- BUT, what if we ask the opposite question?
- Given spatio-temporal numerical data can we determine the underlying differential equation?

$$u_t = vu_x + Du_{xx} + C \qquad \rightleftarrows$$

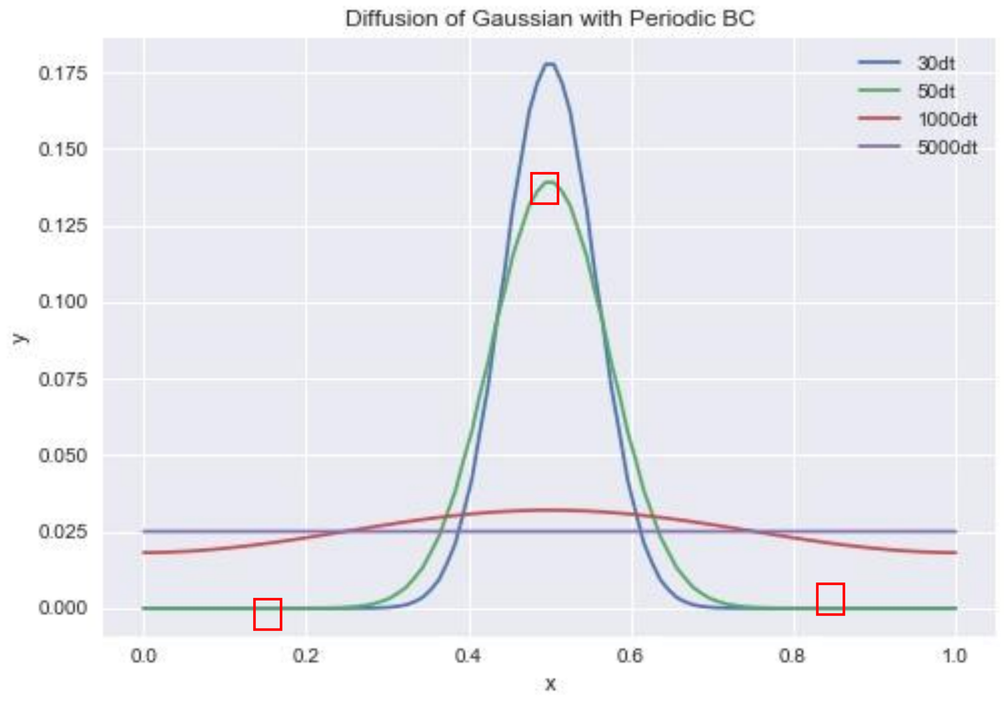


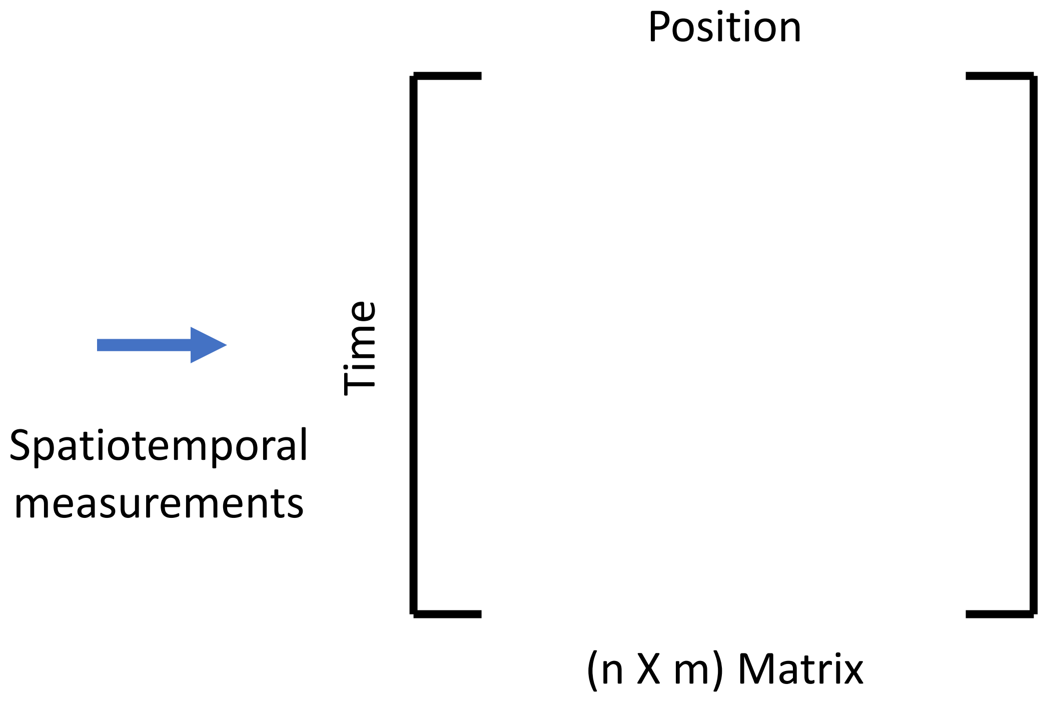
Motivation

- Spatio temporal measurements can be used to quickly determine the governing physics (given certain assumptions)
- Good when first principle derivations are intractable
- Exciting opportunity to solve a big data problem

Current Solutions

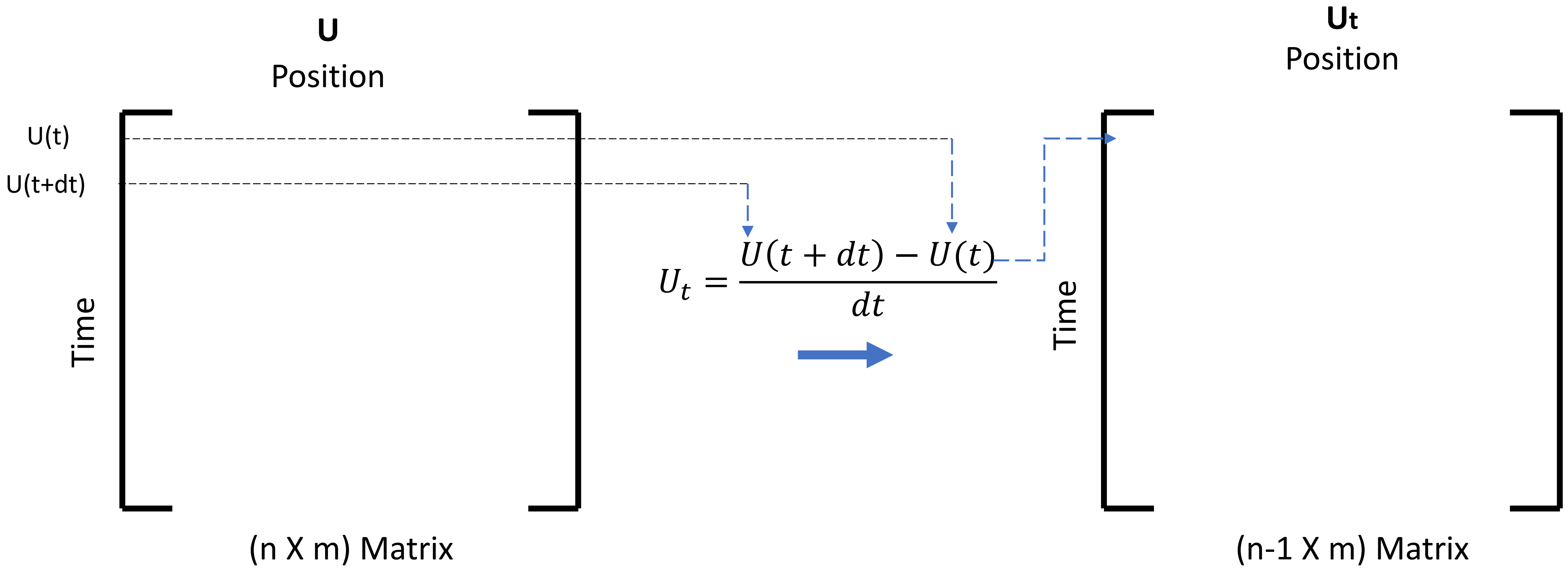
- 1. search through a large combinatorial database of equations (cite)
- 2. machine learning (Rudy et.al) – our focus





Process I – Spatiotemporal Derivatives

- Use matrix to determine time and spatial derivatives



$$U_t = \frac{U(t+dt) - U(t)}{dt}$$

Process II – Column Building

- We can compute a range of different derivatives using this method
- Then we vectorize these matrices (ravel technique) and we have the question in ML form namely:

$$\begin{array}{c} \text{Features}(X) \\ U \quad U_x \quad U_{xx} \quad Uu_x \dots Q \\ \left[\begin{array}{cccccc} \; & \; & \; & \; & \; & \; \\ \; & \; & \; & \; & \; & \; \end{array}\right] \end{array} \begin{array}{c} \text{Coefficients}(\theta) \\ \\ \left[\begin{array}{c} \; \\ \; \end{array}\right] \end{array} = \begin{array}{c} \text{Response}(Y) \\ U_t \\ \left[\begin{array}{c} \; \\ \; \end{array}\right] \end{array}$$

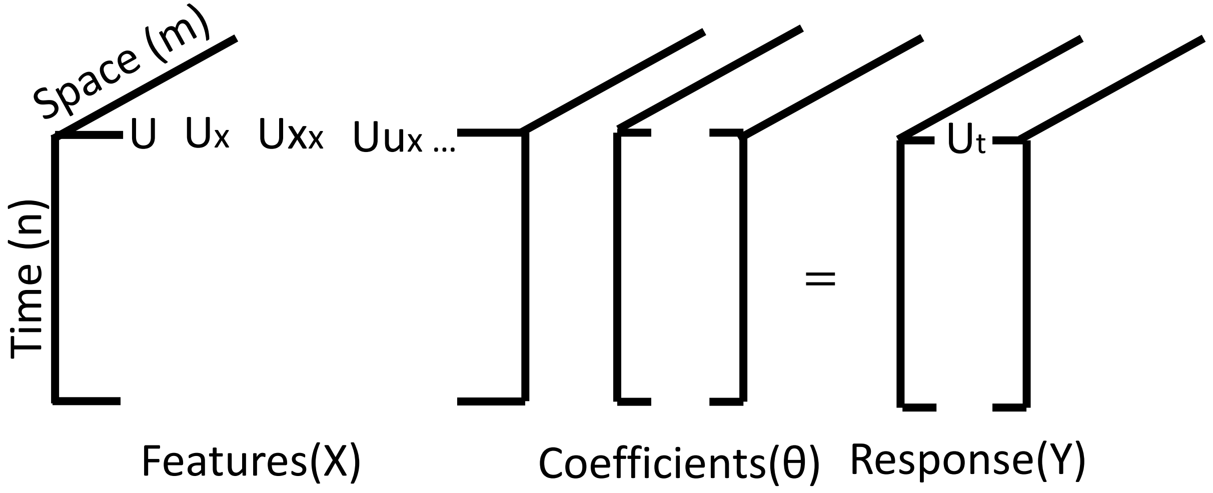
Only difference between this and ML is we don’t have a training/test split.

Note: Q vector can be a potential in SE.

Process III – 2 Algorithms

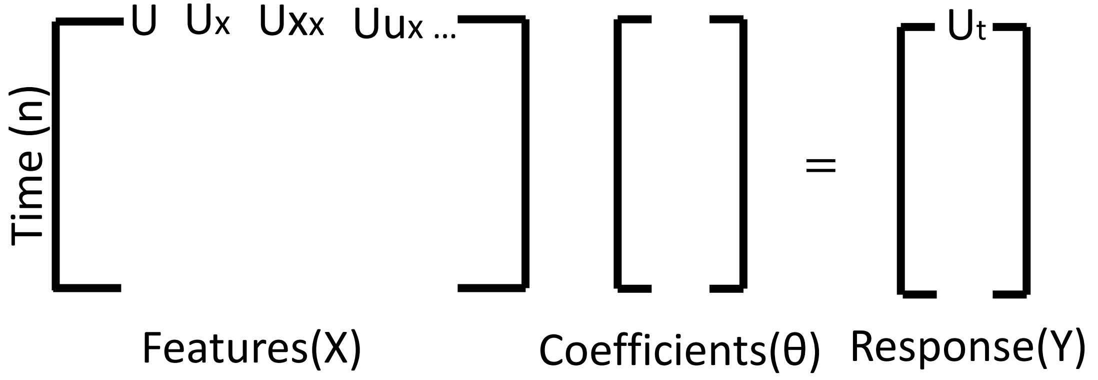
Algorithm 1

Recall that U is a matrix – we can build coefficients for multiple space or time steps and average:



Algorithm 2

We can vectorize U e.g if it is mXn we can turn it into an m*n length vector



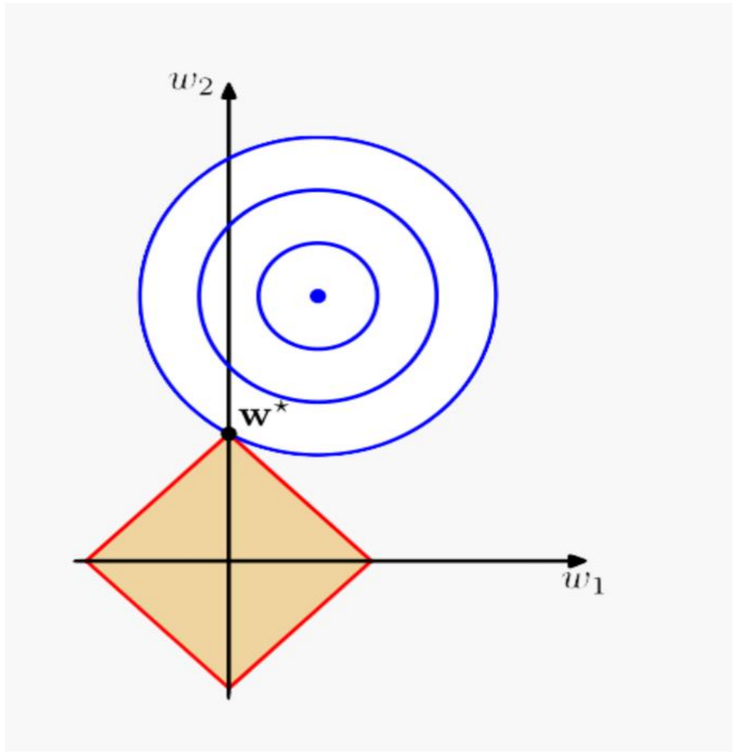
Process IV: Finding Coefficients

- Minimize least squares error + regularization parameter to find theta

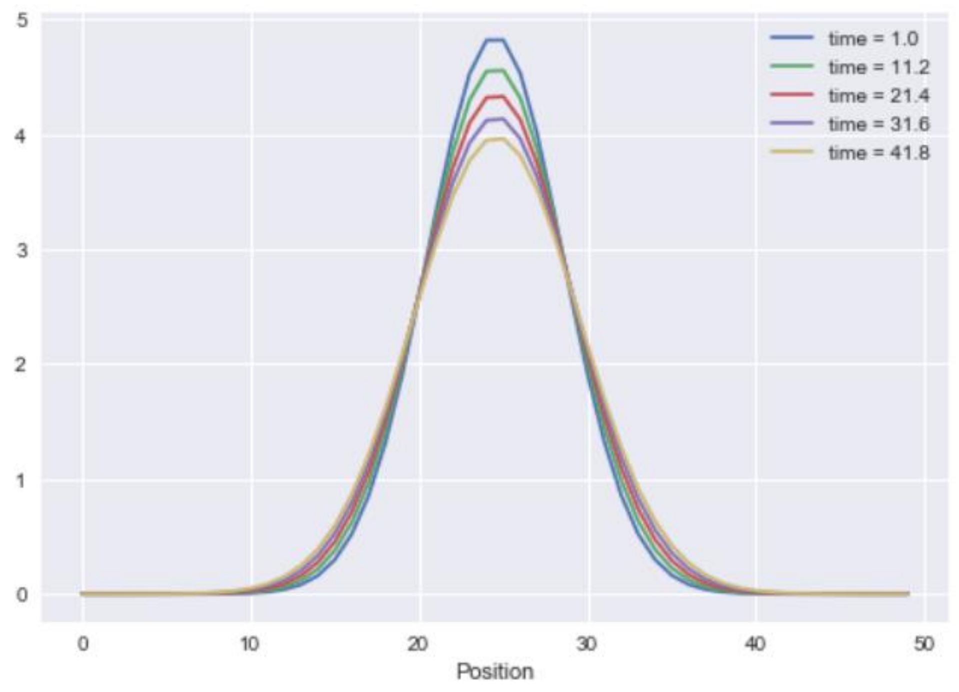
$$cost\ function = \sum(\theta x_i - y_i)^2 + \lambda|\theta|$$

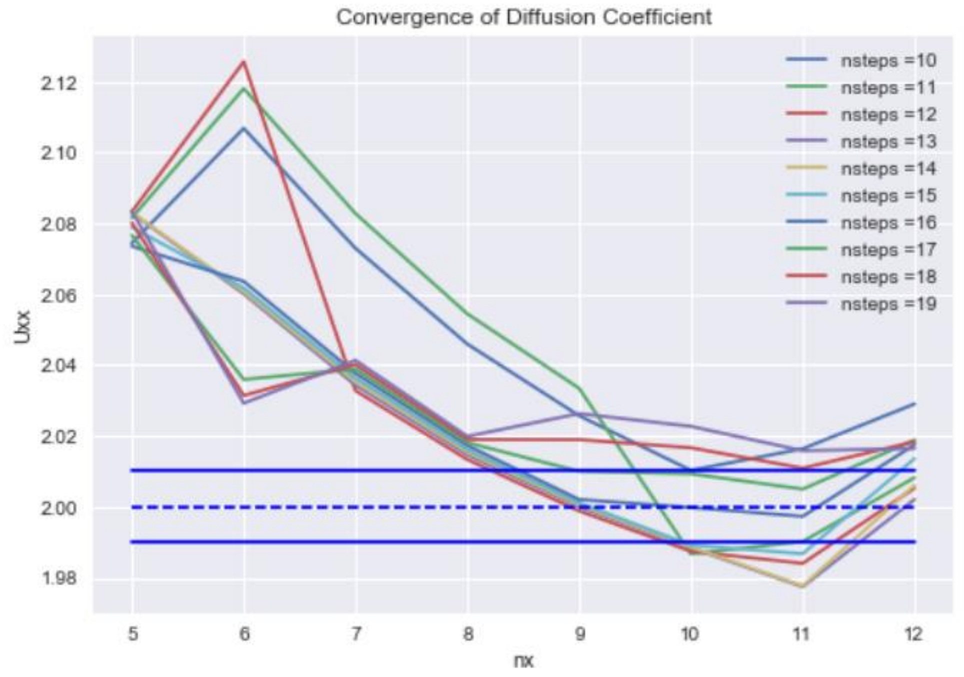
- Additional term (\lambda) -> sparse regression

- Lambda computed through cross validation
  - train/test split randomly without replacement
  - Find lambda that minimizes average cost



Results: Multiple Regression – AD Equation





Multiple regression keeps U matrix in 2D form and computes coefficients for each spatial position across time

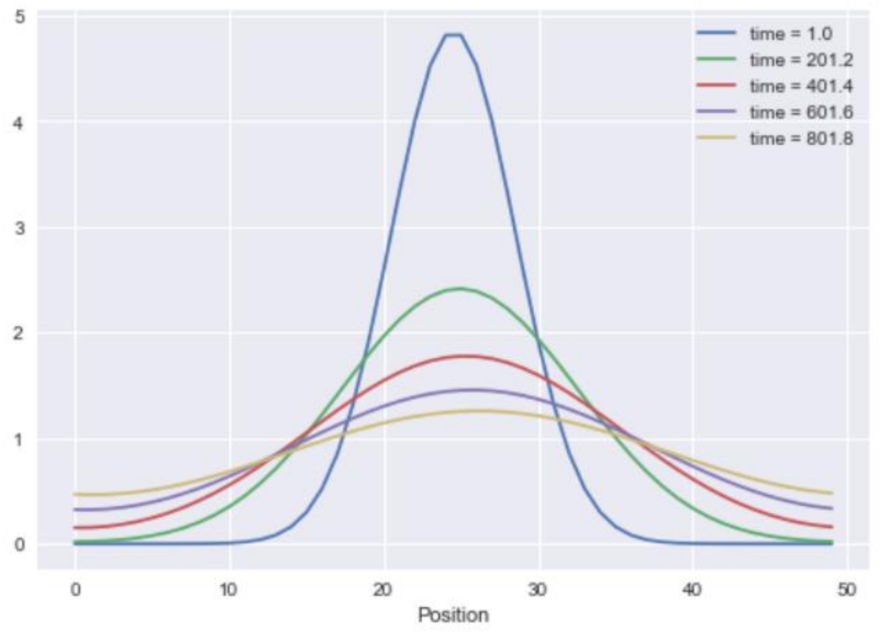
Leads to many coefficient estimates which are then averaged

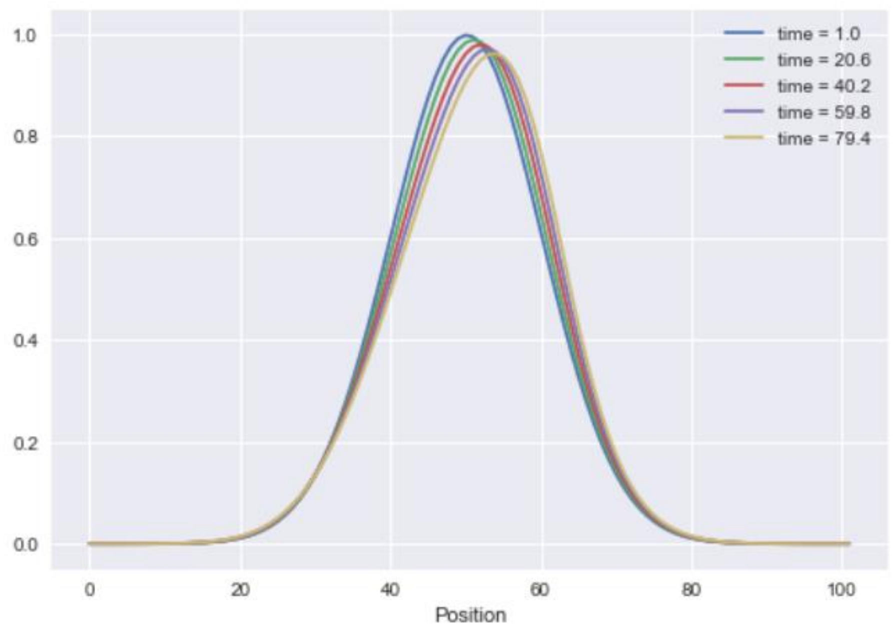
Poor performance, only Uxx terms are well mapped

	numerical	analytic	expected
U	0.000000	0.000000	0.0
U2	0.000000	0.000000	0.0
Ux	-0.026874	0.000000	0.5
Uxx	0.499194	0.495838	0.5
UU2	-0.002466	0.000000	0.0
UUx	-0.005082	-0.024701	0.0

Nx = 50, nsteps = 50

Continued: ADR/Burgers





Coefficients	Obtained	Expected
U	0	1
U^2	0	2
Ux	-0.12	1
Uxx	1.5	1.5
UU^2	0	0
UUx	-0.11	0

Coefficients	Obtained	Expected
U	0.007	0
U^2	0.0019	0
Ux	-0.05	0
Uxx	0.0055	1
UU^2	-0.003	0
UUx	-0.07	1

Algorithm 2 (Rudy et.al) – AD Equation

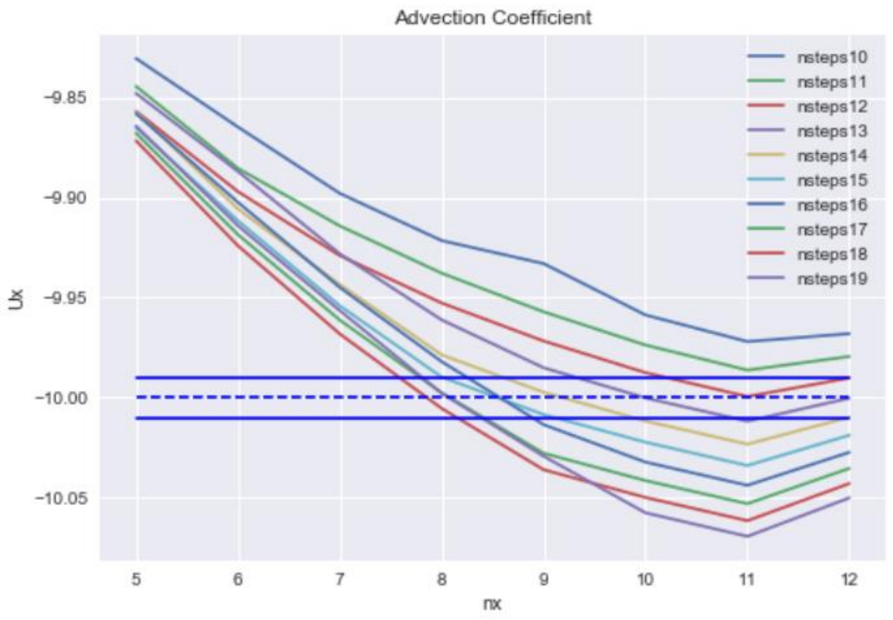
	numerical	analytic	expected
U	-0.000000	0.000000	0
U2	-0.000000	-0.000000	0
Ux	-1.309812	-1.297728	2
Uxx	0.998996	1.003325	1
UU2	-0.000000	-0.000000	0
UUx	-0.216654	-0.225324	0

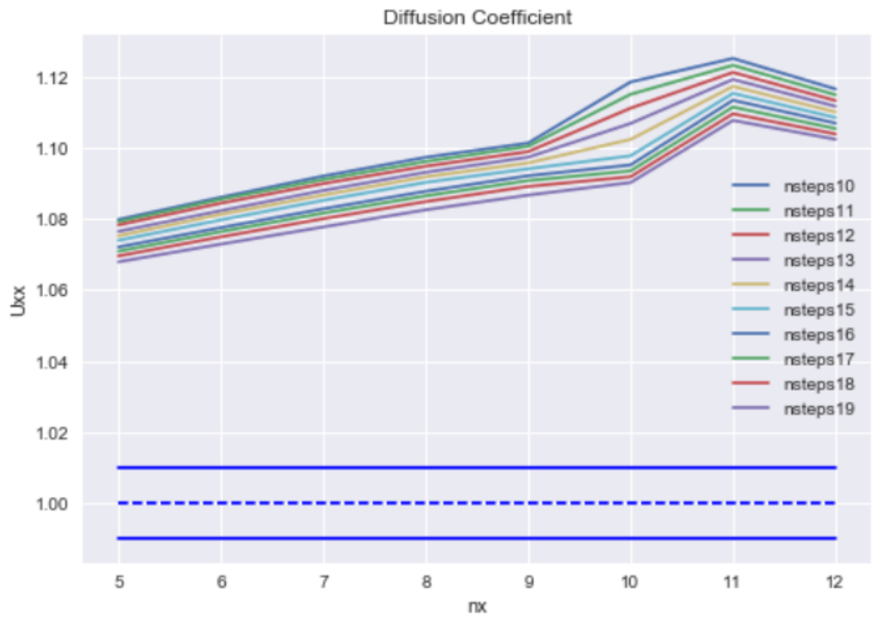
	numerical	analytic	expected
U	-0.000000	-0.000000	0
U2	-0.000000	-0.000000	0
Ux	-3.929435	-5.454719	6
Uxx	2.996989	3.002024	3
UU2	-0.000000	-0.000000	0
UUx	-0.649963	-0.239834	0

	numerical	analytic	expected
U	-0.000000	-0.000000	0
U2	-0.000000	-0.000000	0
Ux	-1.238900	-3.549935	4
Uxx	3.995993	4.000994	4
UU2	-0.000000	-0.000000	0
UUx	-0.866743	-0.192171	0

Nx = 50, nsteps = 50 – same parameters as algo 1 but much better performance for different velocities/diff coef.
Now its worth tuning nx/nsteps to make this better!

Algorithm 2 (Rudy et.al): AD Equation (numerical)





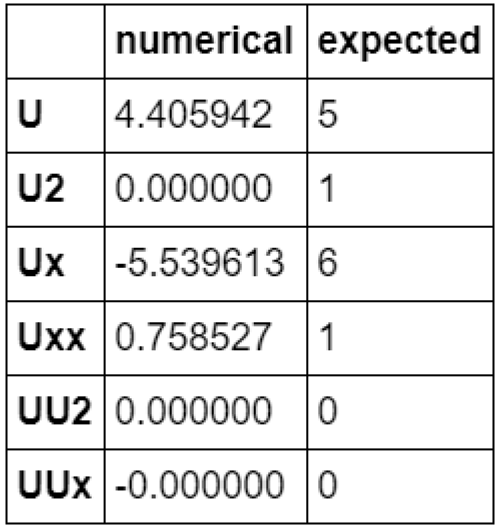
Coefficients	Obtained	Expected
U	0.029	0
U^2	0	0
Ux	-9.7	-10
Uxx	1.07	1
UU^2	0	0
UUx	0	0

Algo 2 works much better – is able to select and determine coefficient terms to a good degree of accuracy.

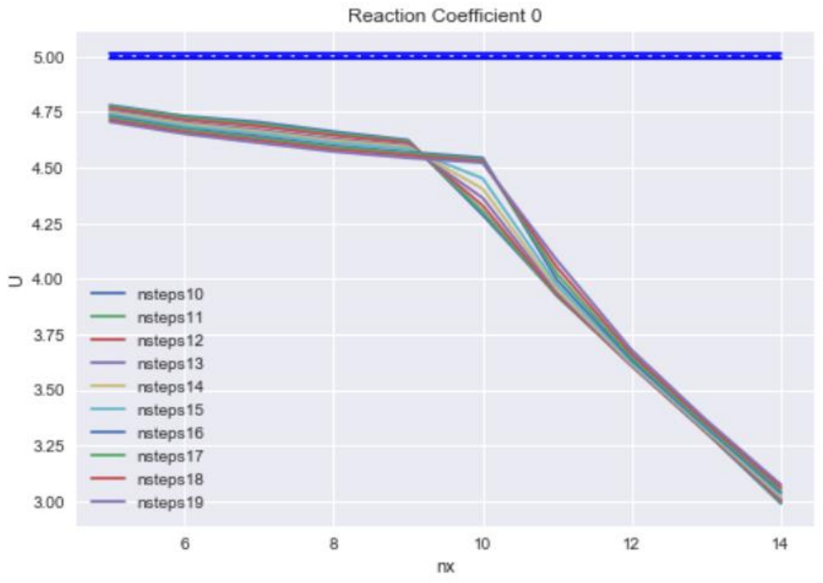
Algo 2: ADR

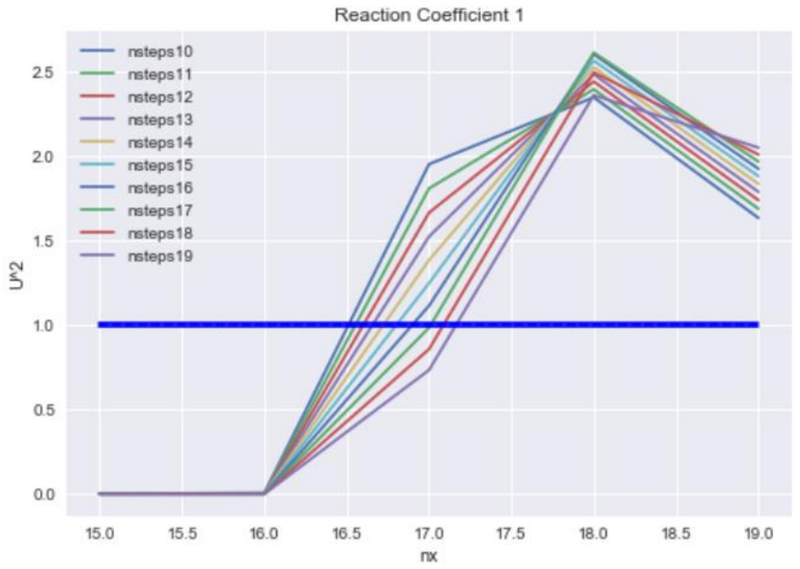
$$u_t = vu_x + du_{xx} + cf(u)$$

ADR: nx = 10, nsteps = 10



	numerical	expected
U	4.405942	5
U2	0.000000	1
Ux	-5.539613	6
Uxx	0.758527	1
UU2	0.000000	0
UUx	-0.000000	0





Reactive terms are still hard to predict with
This algo even with different nx/nsteps

Need nx of 17 for U^2 but nx of 4 for U to be correct.

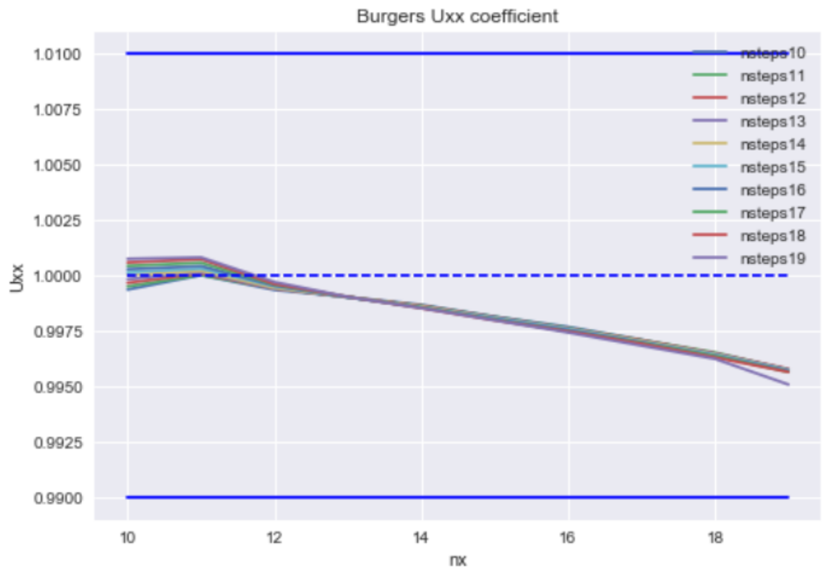
Algo 2: Burgers

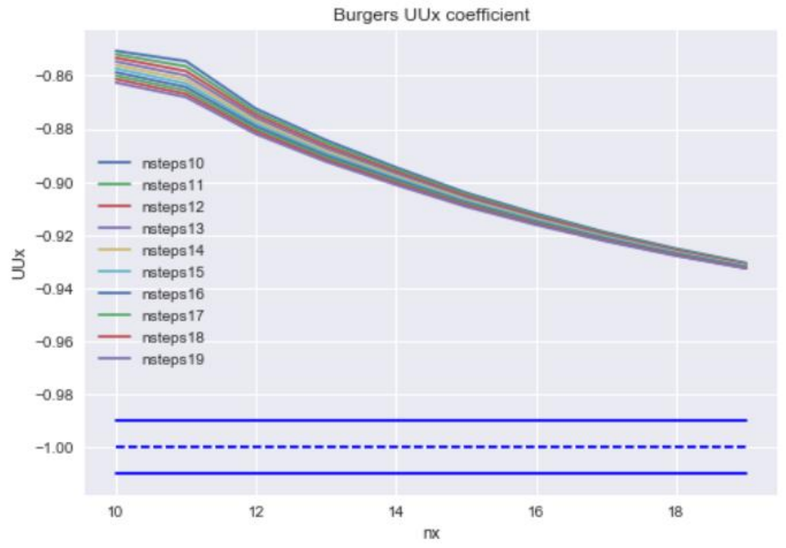
$$u_t = cuu_x + du_{xx}$$

Coefficients	Obtained	Expected
U	0.03	0
U^2	-0.05	0
Ux	-0.016	0
Uxx	0.707	1
UU^2	0.012	0
UUx	-0.967	1

Nx = 50, nsteps = 10

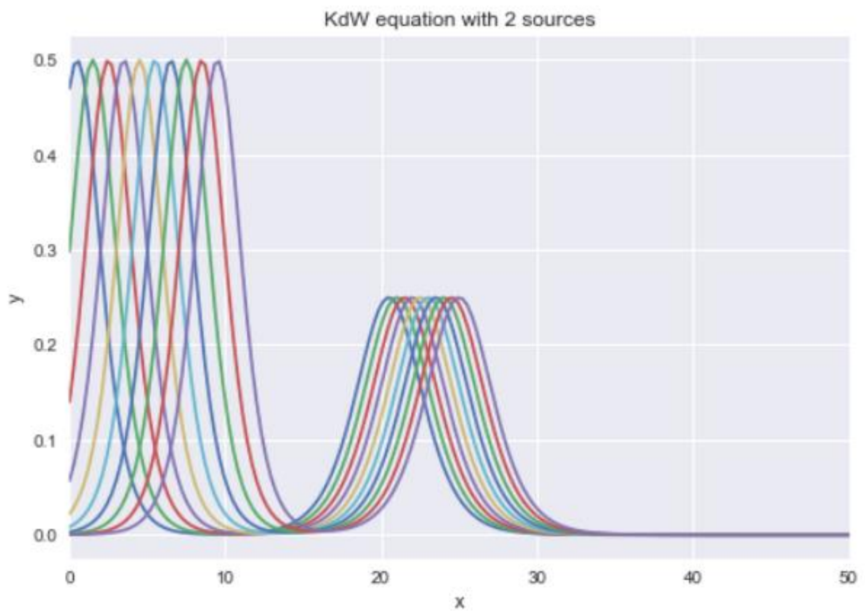
Algo 2 works well here. We do indeed trade off some accuracy in Uxx to gain increase in UUx term





Algo 2: Korteweg de Vries

$$u_t = cuu_x + u_{xxx}$$



Solution in which speed of the wave depends on the height

Hard to find the expected
Tried to run different nx values
Nt issues since we need many nsteps

Coefficients	Obtained	Expected
U	0	0
U^2	0	0
Ux	0	0
Uxx	2.36	0
Uxxx	0	1
UU^2	0.44	0
UUx	96.7	100

Nx =4000, nt =10

Coefficients	Obtained	Expected
U	0	0
U^2	0	0
Ux	0	0
Uxx	3.83	0
Uxxx	0.468	1
UU^2	0.44	0
UUx	98.4	100

Nx =4000, nt =500

Algo 2: QM

$$ih\, u_t = -\frac{h^2}{2m} u_{xx} + \frac{x^2}{2} u$$

Complex function! How do we build a model? Lets use harmonic potential:

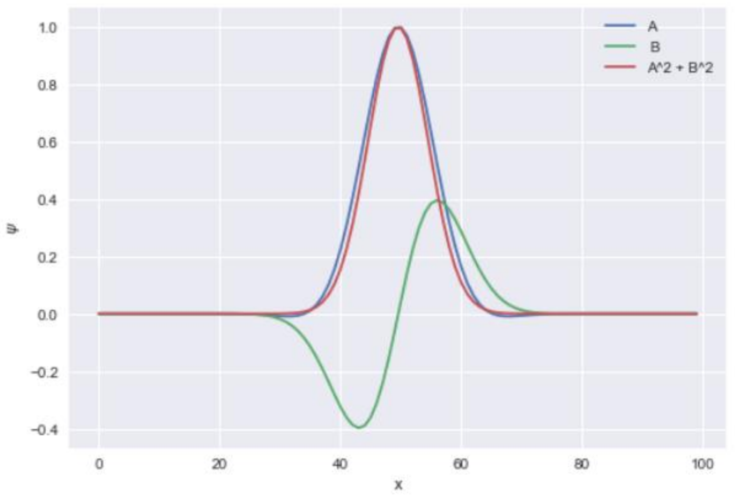
$$ih\, u_t = \frac{h^2}{2m} u_{xx} + 0.5x^2 u$$

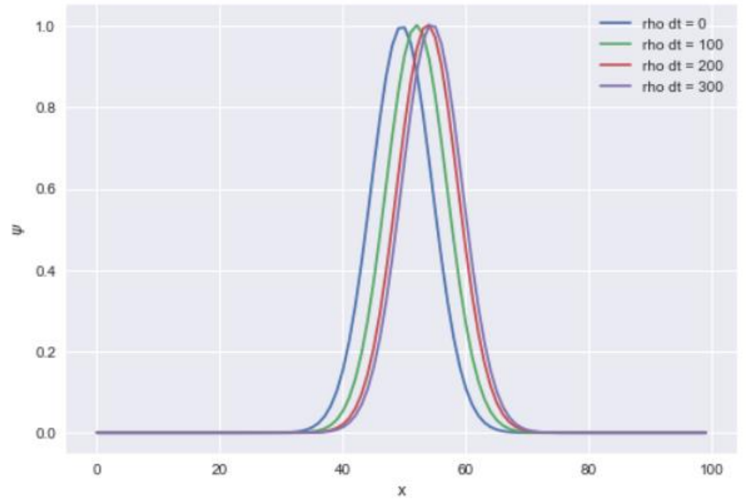
$$u = A + iB$$

$$ih(A_t + iB_t) = \frac{h^2}{2m}(A + iB)_{xx} + 0.5x^2(A + iB)$$

$$-hB_t = \frac{h^2}{2m}(A)_{xx} + 0.5x^2(A)$$

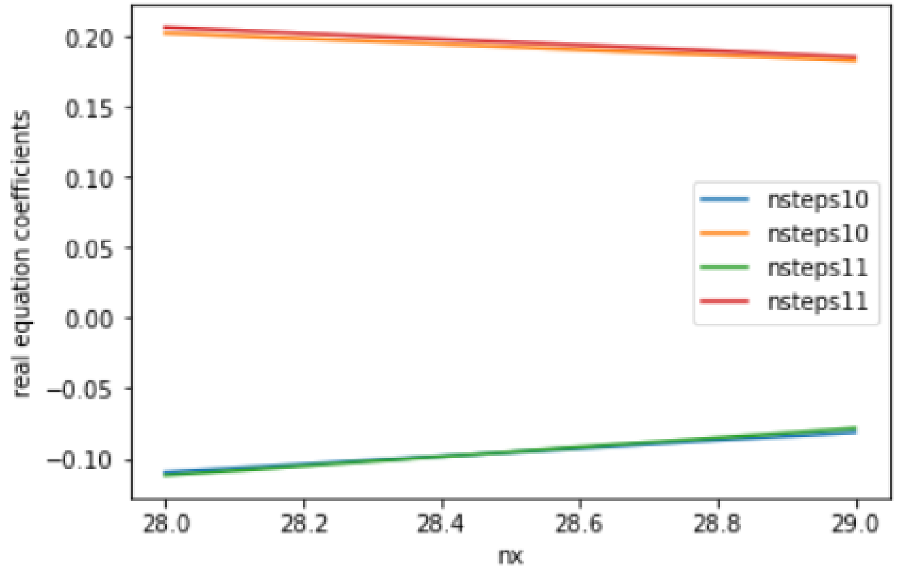
$$hA_t = \frac{h^2}{2m}(B)_{xx} + 0.5x^2(B)$$

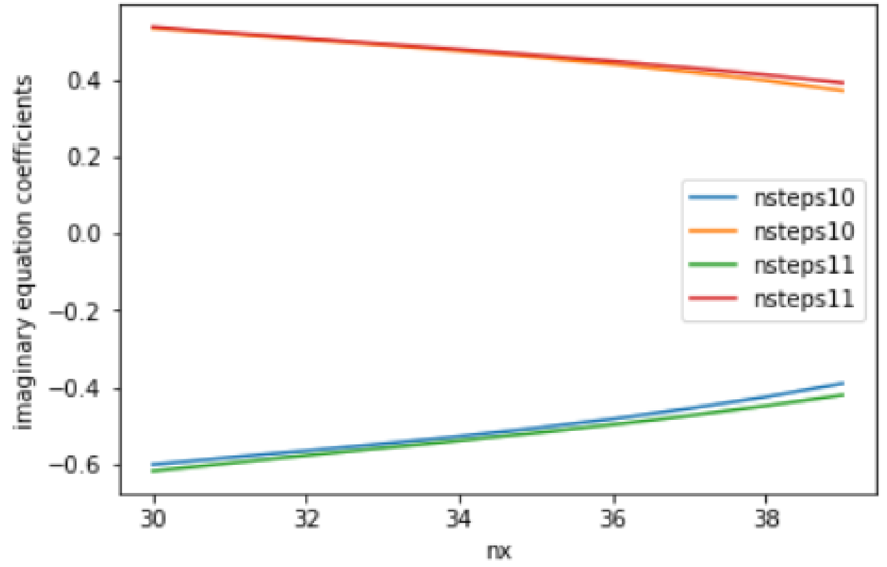




Cannot seem to determine the underlying process – poor coefficient estimates

Algo 2: QM





Coefficients	Obtained	Expected
U	0.13	0
U^2	0.005	0
Ux	0.86	0
Uxx	0.2	1
Uxxx	0.009	0
UU^2	0.11	0
UUx	0.015	0
X^2U	0.11	1

Coefficients	Obtained	Expected
U	1.3	0
U^2	0.2588	0
Ux	1.05	0
Uxx	0.5	1
Uxxx	0.0	0
UU^2	0.0	0
UUx	0.0	0
X^2U	0.6	1

Further Investigation

Raissi et.al. have been able to address a similar problem using a gaussian process approach

Key differences:
1. Raissi assumes we know the general form of the equation so this reduces dimensionality of the problem
2. Assumes only two time steps (reasonable considering our graphs)
3. Tries to address the small data problem e.g. can we determine coefficients with a few datapoints

$$u_t = \lambda_1 u_x + \lambda_2 u_{xx}$$

Idea is to use backward Euler with a small dt. So assume we have measurements at t and t+dt.

$$\frac{(u^n - u^{n-1})}{dt} = \lambda_1 u_x + \lambda_2 u_{xx}$$

$$u^n + dt(\lambda_1 u_x + \lambda_2 u_{xx}) = u^{n-1}$$

Transform the nonlinear operator into a linear one and then set:

$$u^n = GP(0, k(x, x', \theta))$$

Further Investigation Continued

$$k(x,x',\theta) = \gamma^2 \exp(-\frac{1}{2}\sum w_d^2 (x_d - x'_d)^2)$$

$$-\log \mathrm{p}(\mathrm{h}|\theta,\lambda,\sigma^2) = 0.5\mathrm{h}^{\mathrm{T}}\mathrm{K}^{-1}\mathrm{h} + 0.5 \log|\mathrm{K}| + \frac{\mathrm{N}}{2}\log(2\pi)$$

Math aside – key takeaways include:

1. Assume output of linear system whose input is gaussian is gaussian.
2. Minimizing log odds gives us a shot at predicting coefficients.
3. Major assumption of knowing the actual system.

Future Work

1. Test model on higher order derivatives (e.g. Kuramoto-Sivashinsky)
2. Reach out to Rudy et.al. to figure out how SE was solved using algo 2
3. Try other regression techniques (e.g. Ridge/Random Forest)
4. Tackle small data challenge using Gaussian Process (code available on git)
5. Extend to 2-D and 3-D

Conclusion

- 2 approaches of learning have been presented
- Algo 2 works significantly better at detecting PDE’s
    - Ideal performance with AD and Burgers
    - Good performance with ADR (need to check analytic solution)
    - Poor performance with SE and KdW
- Unique aspect is our ability to map back from numerical solution to PDE
- Issue is that we need a specific range of nx/nsteps to make sure this happens well
- Unlike models presented – this works with numerical solvers and not analytic solutions to generate data