EngSci 721

Inverse Problems and Learning From Data

Oliver Maclaren (oliver.maclaren@auckland.ac.nz)

1. *Basic concepts* [5 lectures + 1 Tutorial]

   Forward vs inverse problems. Well-posed vs ill-posed problems. Algebra and calculus of inverse problems (left and right inverses, generalised and pseudo inverses, resolution operators, matrix calculus). Representing higher dimensional problems (image data etc).

2. *Instability and regularisation in linear and nonlinear problems* [6 lectures + 1 Tutorial]

   Instability and related issues for generalised inverses. Introduction to regularisation and trade-offs. Tikhonov regularisation. Higher-order Tikhonov regularisation. Sparsity and regularisation using different norms. Truncated singular value decomposition. Iterative regularisation, including stochastic/mini-batch gradient descent.

3. *Further topics* [3 lectures + 1 Tutorial]

   Regularisation parameter choice, including statistical and machine learning views of regularisation. Confidence sets for linear and nonlinear models. Physics-informed machine learning and neural networks.

Module overview

Inverse Problems and Learning From Data (*Oliver Maclaren*)
[~14 lectures/3 tutorials]

Lecture 8: Sparsity and L1 regularisation

Topics:

- Sparsity and L1 regularisation

Eng Sci 721 : Lecture 8

Regularisation - Beyond Tikhonov? L_1 norm

- L_1 norm
  - Sparsity
  - Formulation as iterated L_2 reg.
  - Total variation
  - Robust regression

Other norms?

'Tikhonov' regularisation generally refers to the L_2 norm for both 'data space' & 'model space' (possibly D)

→ Useful computationally (eg re-frame as standard least squares, differentiable...)

→ But other norms can be useful to emphasise different data &/or model features

mix & match data & model norms

In particular, the L_1 norm is closely related to eg

L_1 regularisation:
- Sparsity/compressed sensing
- Total variation reconstruction

L_1 objective/loss function:
- Robust regression

(The L_0 & L_∞ norms also come up fairly often)

L_1 norm & Lp norms

The L_1 norm has the form

$$\|x\|_1 = |x_1| + |x_2| + \cdots |x_n|$$

Both the L_1 & L_2 norms are special cases of Lp norms:

$$\|x\|_p = \left(|x_1|^p + |x_2|^p + \cdots |x_n|^p\right)^{1/p}$$

for $p \geq 1$.

The L_∞ norm is the limit as $p \to \infty$ and given by

$$\|x\|_\infty = \max\left\{|x_1|, |x_2|, \cdots |x_n|\right\}$$

The L_0 'norm' is not technically a norm but means 'number of non-zero' entries in x

ie $\|x\|_0 = \text{card}(x)$

where 'card' means 'cardinality'.

L_1 & L_0 'norms': Sparsity.

The L_0 'norm' measures 'sparsity'

→ small L_0 'norm' means few non-zero entries } 'sparse'

→ can get 'simple' solutions in sense of 'only a few non zero components'

→ Unfortunately is basically just enumeration & very difficult to work with L_0 directly

> A famous & somewhat surprising result is that the L_1 norm, when used as regularisation also tends to produce sparse solutions, ie solutions with exactly zero elements

Furthermore: (linear) L_1 problems lead to

convex optimisation problems

that can still be solved relatively efficiently

(eg Linear / Quadratic Programming etc)

Equivalence of norms ?

In contrast to L_0/L_1, L_2 produces solutions with very small but non-zero elements

But aren't all norms in $\mathbb{R}^n$ 'equivalent'? ie

$$\exists\, \alpha, \beta \text{ s.t. } \alpha \|x\|_a \leq \|x\|_b \leq \beta \|x\|_a$$

for any two norms $\|\cdot\|_a$ & $\|\cdot\|_b$

→ value of norm can be approximated, but the geometry & solutions favoured by different norms are different qualitatively

↳ also, α, β matter! (& as $n \to \infty$ approx gets worse)

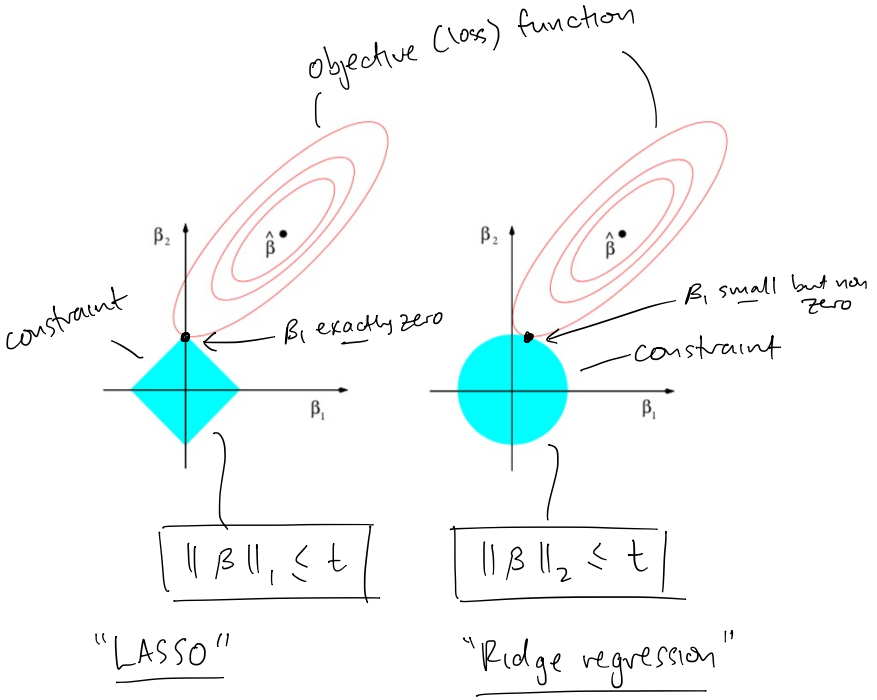
→ quantitatively similar evaluation of given solution but problems lead to qualitatively different solutions as 'best'

L_2 : - lots of small but non-zero values
- related to 'average'

L_0/L_1 : - lots of exactly zero values
- related to 'median'
- more robust/less sensitive to large individuals than L_2.

L_1 vs L_2 regularisation

$\min \quad \|A\beta - y\|_2^2$	$\min \quad \|A\beta - y\|_2^2$
s.t. $\quad \|\beta\|_1 \leq t$	s.t. $\quad \|\beta\|_2 \leq t$



[From 'Statistical learning with sparsity' by Hastie et al.]

Applications of L_1: Sparse solutions

('LASSO' regression)

$$\min \|x\|_1 \quad \text{s.t.} \quad \|Ax - y\|_2 \leq \delta$$

same as before, but emph. model norm.

or variational form:

$$\min \|Ax - y\|_2^2 + \lambda \|x\|_1$$

key form.

→ Tikhonov-like, but L_1 on model/parameters x instead of L_2

→ Non-differentiable at $x = 0$

→ But: convex optimisation when A is linear

↳ use eg { linear/quad. programming, iterative reweighted LS etc

↳ many existing libraries for LASSO!

→ Can also use differentiable approx. to L_1

→ or just derivative-free optimisation etc.

Examples

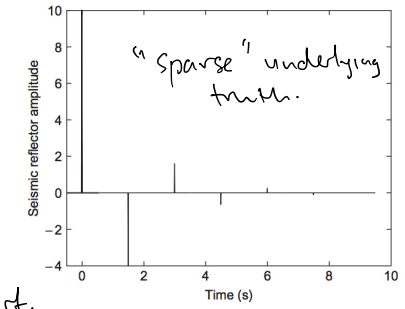
- LASSO regression – Google!
  - Hastie et al
  - Sklearn (python machine learning)

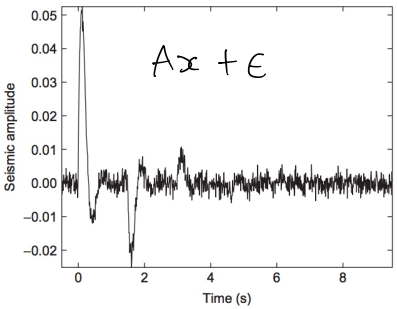
- Aster et al (Inverse Problems)

  Example 7-2 (Full detail attached →)

Deconvolution seismic sensing

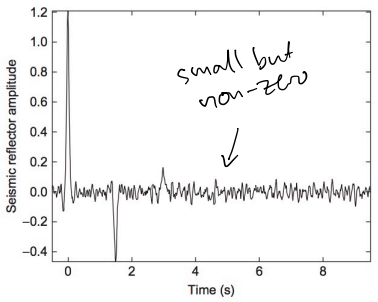
True signal → Observations



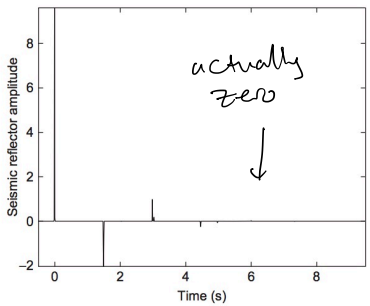


invert.

L_2 regularisation recovered signal:



L_1 regularisation recovered signal:



Applications of L_1 : Total variation regularisation

$$\min \|D_1 x\|_1$$

$$\text{s.t. } \|Ax - y\|_2 \leq \delta$$

or variational form:

$$\boxed{\min \|Ax - y\|_2^2 + \lambda \|D_1 x\|_1}$$

→ Tikhonov-like, but L_1 on model 'roughness' $D_1 x$ instead of L_2

→ piecewise constant for D_1 (penalise towards $D_1 x = 0$)

→ allows for small number of discontinuous 'jumps' in value

↳ good for recovering objects with 'sharp' features like edges in images

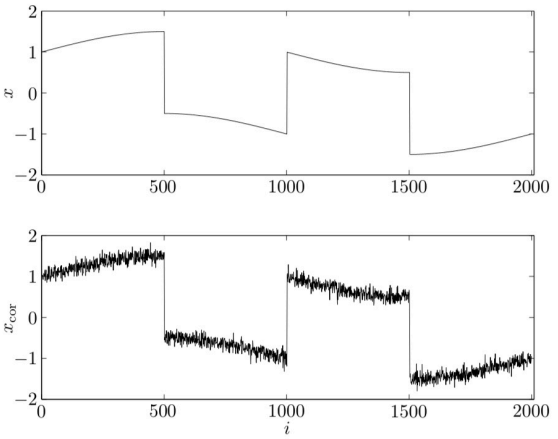
→ piecewise linear for $\|D_2 x\|$ (penalise towards $D_2 x = 0$)

↳ allows for jumps in first derivative

→ Solving: same basic idea as for $D_0 = I$

(convex / LP / QP / IRLS etc)

Example: Boyd & V. 6.3.3



← True signal

← corrupted / noisy signal

Figure 6.11 A signal $x \in \mathbf{R}^{2000}$, and the corrupted signal $x_{\text{cor}} \in \mathbf{R}^{2000}$. The noise is rapidly varying, and the signal is mostly smooth, with a few rapid variations.

$\|Dx\|_2^2$ regularised soln's

$\|Dx\|_1$ regularised soln's

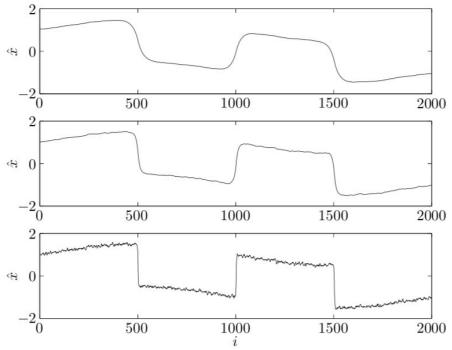


Figure 6.12 Three quadratically smoothed signals $\hat{x}$. The top one corresponds to $\|\hat{x} - x_{\text{cor}}\|_2 = 10$, the middle one to $\|\hat{x} - x_{\text{cor}}\|_2 = 7$, and the bottom one to $\|\hat{x} - x_{\text{cor}}\|_2 = 4$. The top one greatly reduces the noise, but also excessively smooths out the rapid variations in the signal. The bottom smoothed signal does not give enough noise reduction, and still smooths out the rapid variations in the original signal. The middle smoothed signal gives the best compromise, but still smooths out the rapid variations.

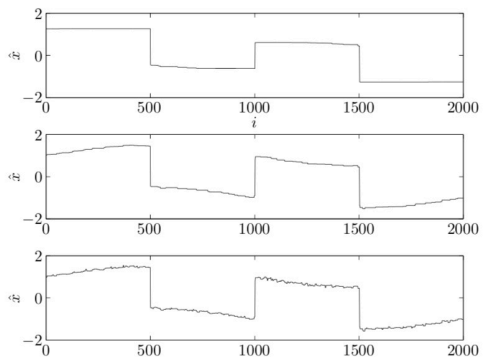


Figure 6.14 Three reconstructed signals $\hat{x}$, using total variation reconstruction. The top one corresponds to $\|D\hat{x}\|_1 = 5$, the middle one to $\|D\hat{x}\|_1 = 8$, and the bottom one to $\|D\hat{x}\|_1 = 10$. The bottom one does not give quite enough noise reduction, while the top one eliminates some of the slowly varying parts of the signal. Note that in total variation reconstruction, unlike quadratic smoothing, the sharp changes in the signal are preserved.

Implementation of L_1 as iterated L_2

- Here is one, relatively simple way to implement L_1 reg. as (iterated L_2 reg.) (many more sophisticated ways to do!)

Note $\|x\|_2^2 = x_1^2 + x_2^2 + x_3^2 + \dots$

& $\|x\|_1 = |x_1| + |x_2| + \dots$

$$\Rightarrow \quad \|x_1\| = |x_1| \cdot \frac{|x_1|}{|x_1|} + |x_2| \cdot \frac{|x_2|}{|x_2|} + \dots$$

$$= \frac{x_1^2}{|x_1|} + \frac{x_2^2}{|x_2|} + \dots$$

$$= w_1 x_1^2 + w_2 x_2^2 + \dots \quad \text{for } w_i = |x_i|$$

Define $W = \begin{bmatrix} w_1 & 0 & \cdots \\ 0 & w_2 & \\ \vdots & & \ddots \end{bmatrix}$ (diagonal with weights)

$$W^{1/2} = \begin{bmatrix} w_1^{1/2} & 0 & 0 \cdots \\ 0 & w_2^{1/2} & \vdots \\ \vdots & \cdots & \ddots \end{bmatrix}$$

$$\Rightarrow \quad \|x\|_1 = \|W^{1/2} x\|_2^2 \quad ! \text{ Tikhonov ??}$$

Not quite! W depends on x

$$\dots = \|W(x)^{1/2} x\|_2^2$$

Iterative (Re-)Weighted Least squares

→ To solve:

$$\min \|W^{1/2}(x)\, x\|^2 \quad (+ \text{other terms..})$$

$$= \min_x \sum_i w_i(x)\, x_i^2$$

for weights depending on x, use:

IR/WLS Algorithm (Iterated weighted Least squares)

0. Set small 'fudge factor' $\delta > 0$
1. Initialise $x \leftarrow x^{(0)}$ for some starting guess for x.

Repeat:

2. For $i = 1 : N$ do

   Update weights $w_i \leftarrow \frac{1}{\max\{|x_i|, \delta\}}$

   End
3. Solve weighted least squares with fixed weights, $\min_x \|W^{1/2} x\|^2$
4. Update x to be soln to 3.

Until: x converged (or max it. reached)

Summary: L_1 as 'pseudo L_2'

Can write

$$\boxed{\|y - Ax\|_2^2 + \lambda \|x\|_1} \leftarrow L_1$$

as

$$\boxed{\|y - Ax\|_2^2 + \lambda \|W(x)\,x\|_2^2} \leftarrow \text{'pseudo } L_2\text{'}$$

for W depending on x. ← key issue.

Not strictly Tikhonov but can solve as 'iterated Tikhonov'

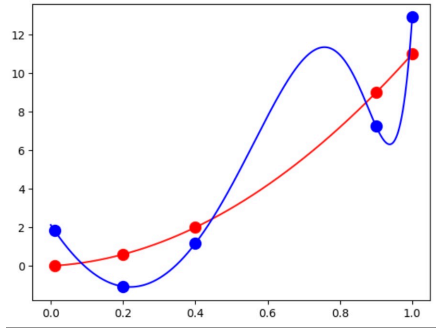
→ Also other methods for solving based on convex optimisation!
- coordinate descent
- linear programming

etc.

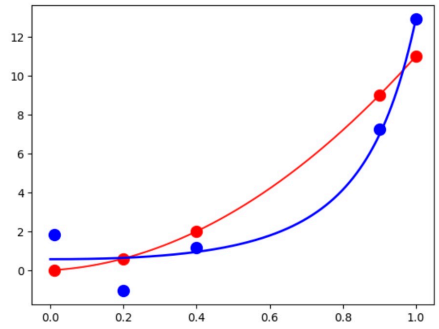
→ also, 'brute force'! (generic nonlinear optimisation prob. solver)

Example: polynomial regression ($\ldots x^{19}$)

no reg:

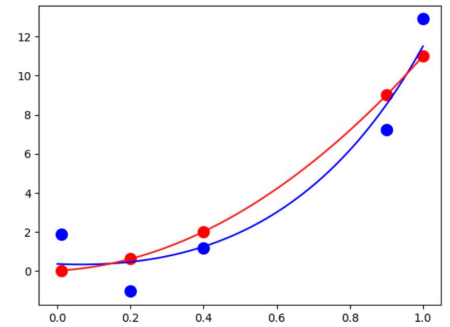


Tikhonov:



L_1:

(via iterated L_2)



Example : resolution?

We can formally write the L_1-inverse as:

$$\left| A^{\ast}_{L_1,\lambda} = (A^T A + \lambda W(\theta))^{-1} A^T \right|$$

However, because $W(\theta)$ depends on θ, this is not a single well-defined matrix (actually solving a nonlinear prob.

→ Get a 'model resol' $A^{\ast}_{L,\lambda}(\theta)\, A$ for each θ! Can look at but ...

→ Easiest to do some 'resolution tests', ie consider

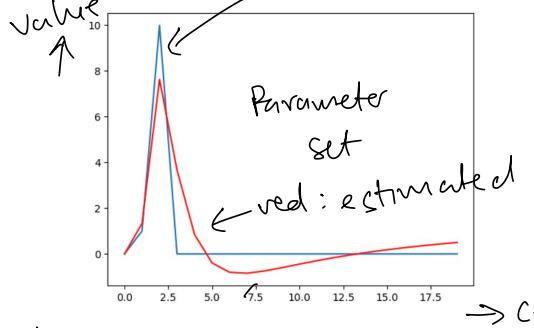
$$\theta^{est,test} = A^{\ast}_{L,\lambda}(\theta^{test})\, A\, \theta^{test}$$

& compare $\theta^{est,test}$ to θ^{test}

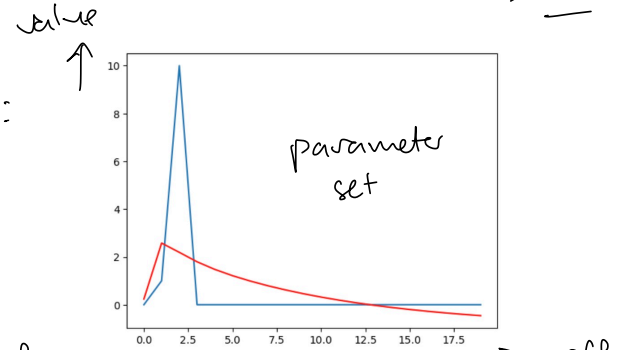
Example : resolution test

blue: true

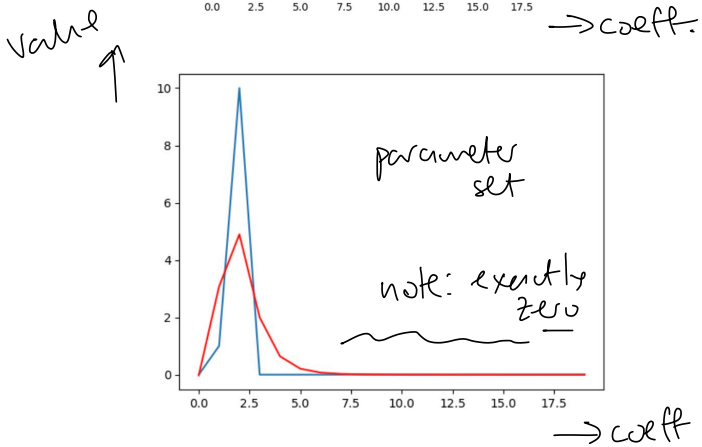
No reg. :



L_2 (Tikhonov):



L_1 :



Other applications of L_1: Robust regression

- Can replace the L_2 norm for data by the L_1 norm

  eg

$$\min \|Ax - y\|_1 + \lambda \|x\|_2^2$$

[Variational/Tikhonov form]

LAD: Least Absolute Deviation regression

- It turns out that, while minimising wrt the L_2 norm leads to the average as the best data fit, minimising wrt the L_1 norm leads to the median as the best data fit

  → This is much more robust to outliers (extreme observations)

  → Not as easy to work with (non-diff) as L_2, though still convex for linear

  ↳ Can eg formulate as a linear programming problem or IRLS prob.

  ↳ Again, many packages exist.

---

simple example of 'robustness'

$$y = (2, 3, 5, 7, 8)$$

$$\text{ave}(y) = 5$$

$$\text{med}(y) = 5$$

$$y' = (2, 3, 5, 7, 80)$$

80: outlier (here: data error?)

$$\text{ave}(y') = 19.4 \gg \text{ave}(y)$$

$$\text{med}(y') = 5 = \text{med}(y)$$

Huber et al 'Robust Statistics':

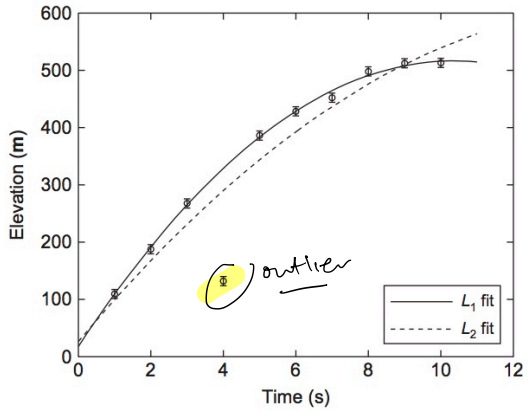
Perhaps the most important purpose of robustness is to safeguard against occcasional gross errors. Correspondingly, most approaches to robustness are based on the following intuitive requirement:

A discordant small minority should never be able to override the evidence of the majority of the observations.

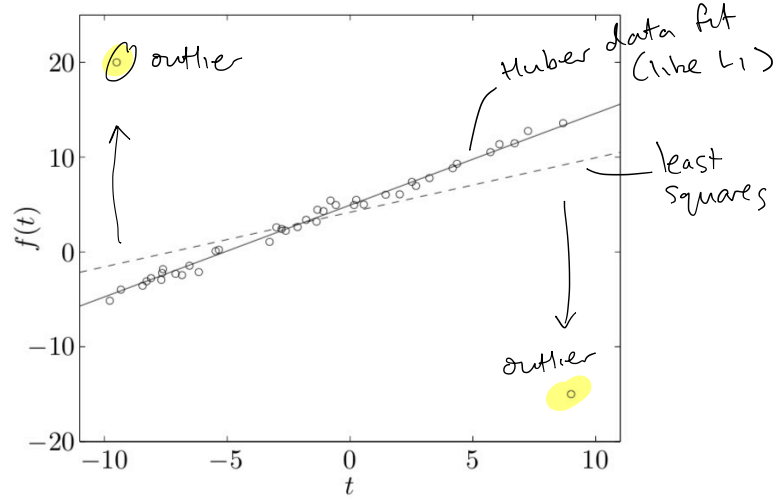
→ Trade-offs! Sensitivity vs stability

Other examples

(Aster et. al Example 2.4)



(Boyd & V. Example 6.2):



Appendix: reformulation of robust regression as linear programming (see eg Boyd & Vandenberghe)

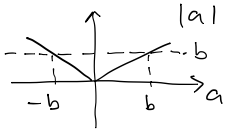
V1: $\boxed{\min \|Ax - y\|_1} = |r_1| + |r_2| + \ldots + |r_m|$

where $|r_i| = \begin{cases} r_i \text{ if } r_i > 0 \\ -r_i \text{ if } r_i < 0 \end{cases}$

→ almost linear (piecewise)

Note: $|a| \leq b$

equiv to $-b \leq a \leq b$



& minimising $|a|$

equiv to $\min_{a,b} b$

note: bilinear. →

st. $|a| \leq b$

ie $-b \leq a \leq b$

Proof? exercise :)

(see also pic ↑)

So we use

V2: $\boxed{\begin{array}{l} \min_{t,x} \; 1^T t + 0^T x \\ \text{st.} \; -t \leq Ax - b \leq t \end{array}}$

Linear programming (linear objective, linear constraints)

Code snippets

```
# L1 reg via iterative L2
thetas = np.zeros((len(alphav),len(thetap_true)))
theta_norms = np.zeros(len(alphav))
data_norms = np.zeros(len(alphav))
I = np.eye(len(thetap_true))
#I[0,0] = 0

for i, alphai in enumerate(alphav):
    theta_current = 0.5*np.ones(len(thetap_true))
    delta = 0.00001
    w = np.zeros(len(thetap_true))
    # solve irls
    nit = 50
    for j in range(0,nit):
        w = 1/np.array([max(delta,abs(thetai)) for thetai in theta_current])
        #print(w)
        #augment
        A_poly_aug = np.vstack([A_poly_obs,alphai*np.diag(w**0.5)])
        #invert
        A_poly_aug_pinv = np.linalg.pinv(A_poly_aug)
        theta_current = A_poly_aug_pinv@yp_aug
    thetas[i,:] = theta_current
    #calc norms
    theta_norms[i] = np.linalg.norm(thetas[i,:],1)
    data_norms[i] = np.linalg.norm(yp_obs - A_poly_obs@thetas[i,:])
```

```
plt.loglog(theta_norms,data_norms**2,linewidth=5)
plt.loglog(theta_norms[nreg],data_norms[nreg]**2,marker='o',markersize=10,linewidth=5,color='k')
plt.ylabel(r'$||A\theta-y||_2^2$',fontsize=14)
plt.xlabel(r'$||\theta||_1$',fontsize=14)
plt.show()
```

✓ 0.5s Python