Homework 9 and 10
APPM 5630 Spring 2023
Advanced Convex Optimization

Due date: Friday, Apr. 7 2023 at midnight (via Gradescope) **Instructor**: Prof. Becker
Theme: Compressed Sensing, sensitivity **Revision date**: 3/23/23

Instructions Collaboration with your fellow students is allowed and in fact recommended, although direct copying is not allowed. The internet is allowed for basic tasks. Please write down the names of the students that you worked with. An arbitrary subset of these questions will be graded.

Homework submission instructions at github.com/stephenbeckr/convex-optimization-class/tree/master/Homeworks. You'll turn in a PDF (either scanned handwritten work, or typed, or a combination of both) to **gradescope**, using the link from the Canvas assignment.

Reading Read the rest of chapter 5 in [BV2004].

Background: compressed sensing

We will introduce compressed sensing along with an example. The basic idea of compressed sensing is that if a signal $x \in \mathbb{R}^n$ is compressible (in particular, assume it only has k nonzero entries), and we acquire it with suitably *incoherent* linear sampling, then we can recover the true signal with about $k \log(n/k)$ measurements, rather than all n samples that we would need for a standard matrix inversion. We will leave the notion of "incoherence" vague for now, but note that the time and frequency domains are roughly incoherent (for the same reason that we have a time-frequency uncertainty principle). For background on compressed sensing, see the review article E. J. Candès and M. Wakin, "An introduction to compressive sampling". *IEEE Signal Processing Magazine*, March 2008.

Let us make a concrete example with an audio signal. We'll take the "Hallelujah" chorus from Handel's *Messiah*, which, conveniently, is included as a .mat file with Matlab. Run[1] `load handel.mat` to load the variables `y` and `Fs`, where $y \in \mathbb{R}^N$ with $N = 73113$, and $Fs = 8192$ meaning the sample rate is 8.192 kHz. According to the Shannon-Nyquist sampling theorem, this means the data contains frequencies up to 4.096 kHz, which is not that high (kids can hear up to about 20 kHz, older adults up to maybe 15 kHz), but still contains most of the notes of interest — for comparison, the note "A" above middle "C", called "A_4", is 440 Hz, and so 4 kHz allows for 3.2 octaves above this.

We can see the frequencies in the signal y in the upper-left plot of Fig. 1. There are two dominant tones, one at about 575 Hz and another at 1.13 kHz. Of course the tones are not static in time, as this is music; see Fig. 2 (left) for the spectrogram of the signal, which shows the spectrum (frequencies) over time.

Suppose we want to store less data, and only sample every-other-entry from y. This creates an effective sample rate of $Fs/2$, leading to a max resolvable frequency of 2 kHz, which does not affect the sound quality much. But if we try to sample every-fourth-entry from y, so the sample rate is $Fs/4$, then the max resolvable frequency is 1024 Hz, which is below our second dominant tone. If we just downsample the signal, we alias the higher-frequencies, and have a new spurious note at 924 Hz, which is the aliased version of the 1.13 kHz note (that is, our new cutoff frequency is $8192/8 = 1024$ and 924 is about the same amount under 1024 as 1130 is over 1024). See the top-right plot of Fig. 1. You can hear the sound of this new signal following the Matlab/python code posted online, and it should sound weird.

[1]scripts for recreating this section in Matlab and Python are on the course github page

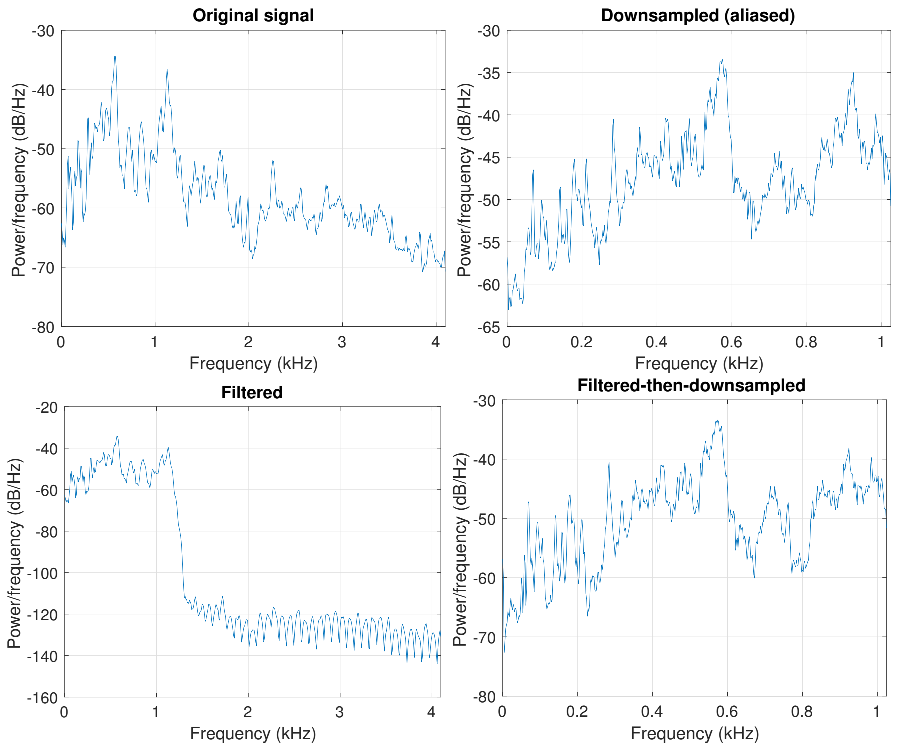

Figure 1: Estimates of the periodogram of the "Handel" signal using the pwelch method with a Modified Bartlett-Hanning window of length 1000.

If we can't hear that highest frequency, it is better to just remove it and not have it alias. The bottom row of Fig. 1 shows (left) a low-pass filtered version of the signal, and (right) the downsampled version of this. We have lost the high notes, but at least there are no weird aliased notes folded in.

Is there any way to recover our signal using only $N/4$ measurements? By looking at the spectrogram in Fig. 2 (left), we see that most entries in the time-frequency plane are negligible (remember, this is in dB, which is a logarithmic scale), so we could set these to zero and have very little effect on sound quality.

More specifically, we are going to consider using version of the modified discrete cosine transform (MDCT). The DCT is like the FFT/DFT in that it maps to a frequency space, but has the advantage of being real-valued. We will take the DCT of blocks of the signal y (we'll use a block-length of 1024), so that we capture the instantaneous note, rather than an average of all notes. We also want to window each block so that we have sharp frequency peaks, but this means we should have the blocks overlap. This is a linear map, and we'll represent it by the symbol D, and it maps $\mathbb{R}^n$ to $\mathbb{R}^{2n}$ (actually, it will map $\mathbb{R}^{73113}$ to $\mathbb{R}^{146226}$ since

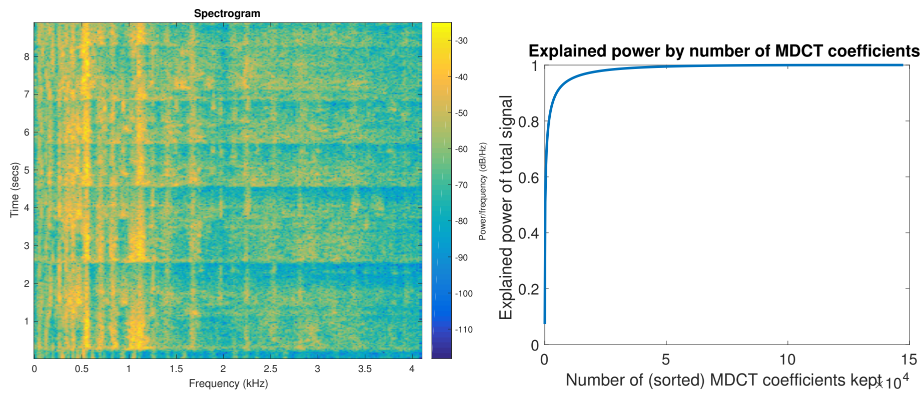

Figure 2: Left: Spectrogram of the "Handel" sound file. In Matlab, this is generated via `load handel.mat; spectrogram(y,5e2,[],[],Fs)`. Right: MDCT coefficients of the Handel signal

we will zero-pad y to be an even multiple of the block size). We use a partition-of-unity window so that $D^T D = I_n$). The code to construct D and D^T is provided on the github page.

If we let $x = D(y)$ be the MDCT coefficients of our audio signal, only a few coefficients of x are large. If we sort them by absolute value, we see in Fig. 2 (right) that we only need to keep a handful of them (much less than a quarter) to faithfully resolve the signal to, say, 90% accuracy.

Compressed sensing takes advantage of the sparsity of x as long as we have incoherent measurements. Also note that since $D^T D = I_n$, that given $x = Dy$, we can recover $y = D^T x$.

Let Ψ be the sampling operator that randomly samples a fourth of all the data, $b = \Psi(y)$. It is important that we take samples at random, and not take exactly every fourth sample, otherwise nothing can be done about the loss of high frequency. Specifically, either give each data point a 1/4 chance of being selected (so the expected length of $\Psi(y)$ is $\frac{1}{4}$ the length of y), or take a random 1/4 of the indices; either way, we draw the random selection once at the very beginning and then fix it — it does not change in subsequent calls to Ψ. Then compressed sensing solves the basis pursuit problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \|x\|_1 \\ \text{subject to} \quad & \Psi D^T x = b \end{aligned} \tag{BP}$$

Solving (BP) is the goal of this week's homework. It is a linear program (LP), but not easy to solve high-dimensional instances of it with usual LP methods since they cannot take advantage of the structure of the MDCT. We will exploit the fact that we can use last week's proximal/projected gradient descent (or FISTA) code to solve a similar least-squares problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \|\underbrace{\Psi D^T}_{A} x - b\|_2 \\ \text{subject to} \quad & \|x\|_1 \le \tau \end{aligned} \tag{LS$_\tau$}$$

The (BP) and (LS$_\tau$) can be made to be equivalent if we choose the right value of τ; in particular, if x_{BP} solves (BP), then we should choose $\tau^\star = \|x_{BP}\|_1$ (but of course we don't know this value). For all $\tau \ge \tau^\star$, the optimal value of (LS$_\tau$) is 0; for all $\tau < \tau^\star$, the optimal value is strictly bigger than 0.

For Homework 9, use (LS$_\tau$), but for actually solving this numerically in Homework 10, use this following equivalent variant which has a smooth objective:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2}\|\Psi D^T x - b\|_2^2 \\ \text{subject to} \quad & \|x\|_1 \le \tau \end{aligned} \tag{LS$_\tau$, ver. 2}$$

$$\boxed{\text{Notation: let } A = \Psi D^T \text{ and } f_0(x) = \|Ax - b\|_2.}$$

Homework 9: derivatives via the dual problem

Read Homework 10 if you want motivation for this section.

Problem 1: Introduce a slack variable z such that $Ax + z = b$, and Lagrange multipliers λ (for the $\|x\|_1 \le \tau$ constraint) and ν (for the $Ax + z = b$ constraint), and write down the Lagrangian and dual function for (LS$_\tau$) (really, for the modified version of (LS$_\tau$) that has the slack variables). Alternatively, use Fenchel-Rockafellar duality (in which case you do not need to introduce z). Either way, the dual function should be found explicitly — it should be something that you could give to CVX/CVXPY to solve.

Problem 2: If $\tau > 0$, can we guarantee strong duality for (LS$_\tau$)?

Problem 3: Let x_τ be the solution to (LS$_\tau$), and let

$$\sigma(\tau) = \|Ax_\tau - b\|_2 \tag{1}$$

be the corresponding value of the objective. Suppose λ_τ and ν_τ are dual optimal variables. What is σ' (that is, $d\sigma/d\tau$) ? *Hint: see section 5.6 in [BV2004].*

Problem 4: a) What are the KKT conditions for (LS_τ)? Simplify them as much as possible.

b) Supposing we are given the primal solution x_τ such that $f_0(x_\tau) > 0$, then use the stationarity KKT condition to find the optimal Lagrange multiplier ν. Then use this to find the optimal Lagrange multiplier λ.

Homework 10: Compressed sensing

You may use the files on the class github website, or your classmates' files from the previous homework.

Problem 1: Solving (LS_τ) or (LS_τ, ver. 2) via (BP)

a) Solve (BP) using CVX/CVXPY, and making the matrix A and b to be whatever you want (but small dimensions, e.g., $A \in \mathbb{R}^{10\times 20}$; we are not yet specifically solving the problem with the Handel data). Call this solution x_{BP}.

Deliverables: In a printed write-up, show your code.

b) Using $\tau = \|x_{BP}\|_1$, solve (LS_τ, ver. 2) using your projected/proximal gradient descent (or FISTA) code from last week's homework (or from the solutions). You will need to have a function that projects onto the ℓ_1 ball, and this is provided to you on the github page. Verify that your solution x_τ agrees with x_{BP} to at least a few digits of accuracy on the same data (A, b) that you used in Problem 1.

Deliverables: Print out relevant snippets of code, and show evidence that your solution agrees with that from Problem 1.

Problem 2: Solving (BP) via (LS_τ, ver. 2)

Now we want to solve (BP) by solving (LS_τ, ver. 2), but unlike problem 1(b), we don't magically know the value of τ.

For a given τ, we can find x_τ by solving (LS_τ, ver. 2), and hence evaluate $\sigma(\tau)$ from Eq. (1). To solve (BP), our goal is to find the smallest τ such that $\sigma(\tau) = 0$. We can solve $\sigma(\tau) = 0$ via Newton's method in 1 dimension:

$$\tau_{k+1} = \tau_k - \sigma(\tau)/\sigma'(\tau).$$

Using the results from Homework 9, and the same data A, b as in Problem 1, solve for the right value of τ using a few iterations of Newton's method (each iteration requires a call to your proximal gradient descent code). Verify that you converge to the same value of τ found in Problem 1, and that your final solution agrees with x_{BP} that you already calculated in Problem 1. [2]

Tip: you may want to allow your Newton solver to pass extra information into the function that calculates σ and σ' (and you definitely want to calculate σ and σ' at the same time); for example, if x_τ is the solution to Eq. (LS_τ, ver. 2) for a given τ, and then you want to solve for a new $\overline{\tau}$, use the old value x_τ as the initial point in your projected/proximal gradient descent code, and it will greatly speed up convergence.

Deliverables: include both a printed write-up of the main code, and any relevant discussion (do you find the right value of τ?)

Problem 3: Scaling it up to a larger problem.

Make the linear operators Ψ and Ψ^T (but do not make them explicit matrices, which would kill performance — this is why we don't just solve (BP) via CVX/CVXPY); you can use the code provided for you on Github. Now, apply the method of Problem 2 using $A = \Psi D^T$ with $b = \Psi(y)$ where y is the Handel audio signal [remember, helper files in Matlab and Python are

[2] This approach was first developed in the paper Probing the Pareto frontier for basis pursuit solutions by E. van den Berg and M. P. Friedlander, SISC 2008.

available on github to implement D and D^T]. Recover x, and set $\hat{y} = D^T(x)$. Play the sound file $\hat{y}$. Have we done a good job recovering the sound?

Deliverables: include relevant parts of code in the write-up, and relevant discussion (how did it sound?)

Problem 4: **Optional**: we took, on average, one fourth of all entries from y. How much lower can you go and still faithfully recover the signal?

Problem 5: **Optional**: another method to solve (BP) is using the Augmented Lagrangian method, so implement this and compare with the result from Problem 1. See Nocedal and Wright's book and their "Penalty and Augmented Lagrangian" chapter. This approach is the same as the "Bregman iteration" approach of Bregman Iterative Algorithms for ℓ_1-Minimization with Applications to Compressed Sensing by Yin et al. 2008.