

# Homework 2

Jimmy Hickey

Due @ 5pm on February 7, 2020

## 1

**Part 1.** We will work through some details on the Hodrick-Prescott (HP) filter for smoothing time series data. Let  $\mathbf{y} \in \mathbb{R}^n$  denote the values of a signal sampled at  $n$  time points. We assume the data has been generated from the model

$$\mathbf{y} = \boldsymbol{\theta} + \mathbf{e}$$

where  $\mathbf{e} \in \mathbb{R}^n$  is a noise vector of i.i.d. zero mean Gaussian random variable and  $\boldsymbol{\theta}$  is a smooth function, in the sense that its derivatives do not take on values that are “too big.” The HP-filter seeks to recover a smooth  $\boldsymbol{\theta}$  by minimizing a penalized negative log-likelihood:

$$\ell(\boldsymbol{\theta}) = \frac{1}{2} \|\mathbf{y} - \boldsymbol{\theta}\|_2^2 + \frac{\lambda}{2} \|\mathbf{D}_n^{(k)} \boldsymbol{\theta}\|_2^2,$$

where  $\lambda$  is a non-negative tuning parameter and  $\mathbf{D}_n^{(k)}$  is the  $k$ th order differencing matrix for a signal of length  $n$ .

$$\mathbf{D}_n^{(1)} = \begin{pmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{pmatrix} \in \mathbb{R}^{n-1 \times n},$$

and  $\mathbf{D}_n^{(k)} = \mathbf{D}_{n-k+1}^{(1)} \mathbf{D}_n^{(k-1)}$ .

## 1

1. Write the gradient and Hessian of  $\ell(\boldsymbol{\theta})$ .

Recall that for vectors  $a$  and  $b$  and matrix  $A$

$$\frac{\partial a^T b}{\partial b} = a, \quad \frac{\partial b^T A b}{\partial b} = (A + A^T)b$$

$$\begin{aligned}
\ell(\theta) &= \frac{1}{2} \|y - \theta\|_2^2 + \frac{\lambda}{2} \|\mathbf{D}_n^{(k)} \theta\|_2^2 \\
&= \frac{1}{2} (y - \theta)^T (y - \theta) - \frac{\lambda}{2} (\mathbf{D}_n^{(k)} \theta)^T (\mathbf{D}_n^{(k)} \theta)^T \\
&= \frac{1}{2} [y^T y - y^T \theta - \theta^T y - \theta^T \theta] + \frac{\lambda}{2} \theta^T (\mathbf{D}_n^{(k)})^T \mathbf{D}_n^{(k)} \theta \\
&= \frac{1}{2} [y^T y - y^T \theta - y^T \theta - \theta^T \theta] + \frac{\lambda}{2} \theta^T (\mathbf{D}_n^{(k)})^T \mathbf{D}_n^{(k)} \theta \quad y^T \theta \text{ is a scalar}
\end{aligned}$$

$$\begin{aligned}
\nabla \ell(\theta) &= \frac{1}{2} (0 - 2y - 2\theta) + \frac{\lambda}{2} [(\mathbf{D}_n^{(k)})^T \mathbf{D}_n^{(k)} + ((\mathbf{D}_n^{(k)})^T \mathbf{D}_n^{(k)})^T] \theta \\
&= -y - \theta + \lambda (\mathbf{D}_n^{(k)})^T \mathbf{D}_n^{(k)} \theta
\end{aligned}$$

$$\begin{aligned}
\nabla^2 \ell(\theta) &= 0 + I + \lambda ((\mathbf{D}_n^{(k)})^T \mathbf{D}_n^{(k)})^T \\
&= I + \lambda (\mathbf{D}_n^{(k)})^T \mathbf{D}_n^{(k)}
\end{aligned}$$

**2**

2. What is the computational complexity for calculating the gradient and Hessian of  $\ell(\theta)$ ? Be sure to take into account the sparsity in  $\mathbf{D}_n^{(k)}$ .

**3**

3. Prove that  $\ell(\theta)$  is strongly convex.

We will use the condition

$$\nabla^2 f(x) \succeq mI$$

**4**

4. Prove that  $\ell(\theta)$  is  $L$ -Lipschitz differentiable with  $L = 1 + \lambda \|\mathbf{D}_n^{(k)}\|_{\text{op}}^2$ .

**5**

5. Prove that  $\ell(\theta)$  has a unique global minimizer for all  $\lambda \geq 0$ .

## 2

### Part 2. Gradient Descent

You will next add an implementation of gradient descent to your R package. Your function will include using both a fixed step-size as well as one chosen by backtracking.

Please complete the following steps.

**Step 0:** Make an R package entitled “unityidST790”.

**Step 1:** Write a function “gradient\_step.”

```
## Gradient Step
#'
#' @param gradf handle to function that returns gradient of objective function
#' @param x current parameter estimate
#' @param t step-size
#' @export
gradient_step <- function(gradf, x, t) {
}

```

Your function should return  $\mathbf{x}^+ = \mathbf{x} - t\nabla f(\mathbf{x})$ .

**Step 2:** Write a function “gradient\_descent\_fixed.” Your algorithm can stop iterating once the relative change in the objective function drops below `tol`.

```
## Gradient Descent (Fixed Step-Size)
#'
#' @param fx handle to function that returns objective function values
#' @param gradf handle to function that returns gradient of objective function
#' @param x0 initial parameter estimate
#' @param t step-size
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
#' @export
gradient_descent_fixed <- function(fx, gradf, x0, t, max_iter=1e2, tol=1e-3) {
}

```

Your function should return

- The final iterate value
- The objective function values
- The 2-norm of the gradient values
- The relative change in the function values
- The relative change in the iterate values

**Step 3:** Write a function “backtrack.”

```
## Backtracking
#'
#' @param fx handle to function that returns objective function values
#' @param x current parameter estimate
#' @param t current step-size
#' @param df the value of the gradient of objective function evaluated at the current x
#' @param alpha the backtracking parameter
#' @param beta the decrementing multiplier

```

```
#' @export
backtrack <- function(fx, x, t, df, alpha=0.5, beta=0.9) {

}
```

Your function should return the selected step-size.

**Step 4:** Write a function “gradient\_descent\_backtrack” that performs gradient descent using backtracking. Your algorithm can stop iterating once the relative change in the objective function drops below `tol`.

```
#' Gradient Descent (Backtracking Step-Size)
#'
#' @param fx handle to function that returns objective function values
#' @param gradf handle to function that returns gradient of objective function
#' @param x0 initial parameter estimate
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
#' @export
gradient_descent_backtrack <- function(fx, gradf, x0, max_iter=1e2, tol=1e-3) {

}
```

Your function should return

- The final iterate value
- The objective function values
- The 2-norm of the gradient values
- The relative change in the function values
- The relative change in the iterate values

**Step 5:** Write a function “gradient\_descent” that is a wrapper function for “gradient\_descent\_fixed” and “gradient\_descent\_backtrack.” The default should be to use the backtracking.

```
#' Gradient Descent
#'
#' @param fx handle to function that returns objective function values
#' @param gradf handle to function that returns gradient of objective function
#' @param x0 initial parameter estimate
#' @param t step-size
#' @param max_iter maximum number of iterations
#' @param tol convergence tolerance
#' @export
gradient_descent <- function(fx, gradf, x0, t=NULL, max_iter=1e2, tol=1e-3) {

}
```

Your function should return

- The final iterate value
- The objective function values
- The 2-norm of the gradient values
- The relative change in the function values
- The relative change in the iterate values

**Step 6:** Write a function to compute the  $k$ th order differencing matrix  $\mathbf{D}_n^{(k)}$ . Use the Matrix package by adding it to the dependency list in the DESCRIPTION file. Among other things, the Matrix package provides efficient storage and multiplication for sparse matrices.

```
#' Compute kth order differencing matrix
#'
#' @param k order of the differencing matrix
#' @param n Number of time points
#' @export
myGetDkn <- function(k, n) {
}

```

**Step 7:** Write functions 'fx\_hp' and 'gradf\_hp' to perform HP-filtering.

```
#' Objective Function for HP-filtering
#'
#' @param y response
#' @param theta regression coefficient vector
#' @param Dkn sparse differencing matrix
#' @param lambda regularization parameter
#' @export
fx_hp <- function(y, theta, Dkn, lambda=0) {
}

#' Gradient for HP-filtering
#'
#' @param y response
#' @param theta regression coefficient vector
#' @param Dkn sparse differencing matrix
#' @param lambda regularization parameter
#' @export
gradf_hp <- function(y, theta, Dkn, lambda=0) {
}

```

**Step 8:** Perform HP-filtering (with  $\lambda = 100$ ) on the following data example using the fixed step-size. Use your answers to Part 1 to choose an appropriate fixed step-size. Try using  $\mathbf{0}$  and  $\mathbf{y}$  as initial values for  $\boldsymbol{\theta}$ . Plot the difference  $\ell(\boldsymbol{\theta}_m) - \ell(\boldsymbol{\theta}_{1000})$  versus the iteration  $m$ . Comment on the shape of the plot given what you know about the iteration complexity of gradient descent with a fixed step size.

```
set.seed(12345)
n <- 1e2
x <- seq(0, 5, length.out=n)
y <- sin(pi*x) + x + 0.5*rnorm(n)

```

- Also plot the noisy data, as points, and smoothed estimates, as a line.

**Step 9:** Perform HP-filtering (with  $\lambda = 100$ ) on the simulated data above using backtracking. Try using  $\mathbf{0}$  and  $\mathbf{y}$  as initial values for  $\boldsymbol{\theta}$ . Plot the difference  $\ell(\boldsymbol{\theta}_m) - \ell(\boldsymbol{\theta}_{1000})$  versus the iteration  $m$ . Comment on the shape of the plot given what you know about the iteration complexity of gradient descent with backtracking.

**Step 10:** Use your code above to smooth some interesting time series data. For example, you might use the tseries R package on CRAN (see the function `get.hist.quote`) to download historical financial data for the daily closing prices of Apple stock over the past two years. Try at least 3 different  $\lambda$  values - different enough to generate noticeably different smoothed estimates - and at least two differencing matrix orders, e.g.  $\mathbf{D}^{(2)}$

and  $\mathbf{D}^{(3)}$ . For all six  $\lambda$  and differencing matrices combinations, plot the noisy data, as points, and smoothed estimates, as a line.