CS170A — Mathematical Modeling & Methods for Computer Science — Fall 2014
HW#4 *Optimization and Numerical Integration*
Due: 11:55pm Wednesday December 10, 2014

D. Stott Parker, Luis Angel Larios Cardenas
stott@cs.ucla.edu, langel@cs.ucla.edu

Using CCLE, please turn in a *FirstName_LastName.zip* archive with your Matlab code and resulting output (enough to show that the program is working) as a document, such as a MSWord file or PDF document, plus any other solution file requested in the following questions.

1. **Income Tax (10 points)**
   You can get income tax returns from any presidential candidate at
   http://www.taxhistory.org/www/website.nsf/Web/PresidentialTaxReturns?OpenDocument
   Some of these returns are amazing – Mitt Romney's 2011 tax return is 379 pages long!

   In this assignment, take the file `M_Romney_2011_tax_return_numbers.txt` of numbers in this incredible tax return and determine (using the method in the course notes) whether the *unique* numbers entered in this form (please omit duplicates) violate Benford's Law.

2. **A Particle System (35 points)**
   In this exercise you will create a particle system by using *Forward Euler Integration* and *Linear Search Optimization*. As we saw in class, *Forward Euler Integration* is the simplest method to integrate forces acting on a body into acceleration, acceleration into velocity, and velocity into a new position. Your objective is to simulate particles in free fall which will bounce off a sinusoidal ground. In order to give your simulation a modest level of realism, you also have to prevent the particles from falling and drifting below the ground by "pushing" them back to their *closest location* on the sine curve. The next steps will help you carry out the simulation of your particle system:

   (a) You will find a `particleSystemTemplate.m` Matlab script within the homework *.zip file that we provided. The script is well-documented and guides you throught the process to complete your simulation. First, locate the **THREE** *%TODO:* sections in that file, enclosed in *%****** blocks. Then, read the file thoroughly and try to understand all the things it does to set up the environment and prepare the simulation variables. What is the timestep δt? Which forces are acting on the particles and what is their *vector representation*?

   (b) You have to simulate at least *5 particles*, whose initial positions and velocities should be randomly generated. So, modify the first *%TODO:* section and write code to give each particle two 2×1 random vectors representing **position** and **velocity**, respectively. Notice that the position random vector components should be constrained to $minX \leq x \leq maxX$ and $minY \leq y \leq maxY$, and that the velocity random vector components should be as $minV \leq x_v, y_v \leq maxV$. Additionally, you have to give each particle a random color by storing a 3×1 vector of RGB triplets, where $0 \leq R, G, B \leq 1$.

   (c) Go to the second *%TODO:* section, which lives within the inner *particles* loop. For the I^{th} particle, you have to update its velocity and position by using the *Euler Forward Integration* method. So, complete the **two** following *assignment* statements:

   ```
   particles(I).velocity = ?
   particles(I).position = ?
   ```

   To simplify things, we have considered that the *mass* of each particle is 1. Therefore, you do not need to integrate force to get acceleration. Instead, use *gravity* directly to obtain the new *velocity*.

   (d) The new particle position you just computed is not quite valid yet; you have to check if the particle has penetrated the sinusoidal ground. If so, you should *translate* the particle to its **closest** point *on* the sine curve. Likewise, the velocity vector should be flipped and scaled so that we have a "bouncing" effect. We have taken care of the latter – but you still have to solve for the position of a penetrating particle. To do so, locate the third *%TODO:* section within the conditional statement

   ```
   if( particles(I).position(2) < sin( particles(I).position(1) ) )
   ...
   ```

here, we are already checking that the particle is indeed *below* the ground. Your goal is to find the point on the sine curve that is the **closest** to the current particle position, so that you can *move* the particle to that safe location (e.g. not penetrating). Let

$$g(x) = \|\mathbf{f}(x) - \mathbf{p}\|^2 = \langle \mathbf{f}(x) - \mathbf{p}, \mathbf{f}(x) - \mathbf{p} \rangle$$

where $\mathbf{p} = \begin{pmatrix} p_x \\ p_y \end{pmatrix}$ is the current particle position, and $\mathbf{f}(x) = \begin{pmatrix} x \\ \sin(x) \end{pmatrix}$ is a vector function for the sinusoidal ground. You have to find the x value such that the distance function $g(x)$ is minimized. How do you minimize $g(x)$?

Use the Matlab builtin function `fzero()` to find the *root* of the expression that minimizes $g(x)$. Name the minimizer x value as $xSol$ — and that's it! the rest of the code will take it from there by fixing the position of a penetrating particle from $\mathbf{p}$ to $\mathbf{p}_{fix} = \begin{pmatrix} xSol \\ \sin(xSol) \end{pmatrix}$.

(e) Finally, record a 10-simulation-second *.avi movie and attach it to your submission file. If your simulation is correct, you should expect an output similar to the clip included in the homework *.zip file.

3. **Regularization (25 points)**

In the traditional least squares model $\mathbf{y} = X\mathbf{c}$, the vector of coefficients $\mathbf{c}$ is often chosen to minimize the least squared error

$$\epsilon(\mathbf{c}) \;=\; \|\mathbf{y} \;-\; X\mathbf{c}\|^2.$$

(a) We can add a *constraint* that requires $\|\mathbf{c}\|^2$ to be small. This is often implemented by adding a *regularization term* $R(\mathbf{c})$ to the error that is very large for values of $\mathbf{c}$ that violate the constraints. In this case the iteration seeks to minimize the *regularized problem*

$$\epsilon(\mathbf{c}) \;+\; R(\mathbf{c}) \;=\; \|\mathbf{y} \;-\; X\mathbf{c}\|^2 \;+\; R(\mathbf{c}).$$

Tikhonov regularization $R(\mathbf{c}) = \|T\,\mathbf{c}\|^2$ uses a covariance matrix T that is chosen to scale the $\mathbf{c}$ values properly. In this case we want to solve the Tikhonov regularized least squares problem, minimizing

$$L(\mathbf{c}, T) \;=\; \epsilon(\mathbf{c}) \;+\; R(\mathbf{c}) \;=\; \|\mathbf{y} \;-\; X\mathbf{c}\|^2 \;+\; \|T\,\mathbf{c}\|^2 \;=\; \left\| \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} \;-\; \begin{pmatrix} X \\ T \end{pmatrix} \mathbf{c} \right\|^2.$$

Give the *normal equations* for this problem, *pseudoinverse*, and formula for the *least squares solution* $\mathbf{c}$.

(b) *Ridge Regression* is a popular form of this least squares problem when $T'\,T = \lambda\,I$ for some $\lambda \geq 0$ (so that $T'\,T$ has a 'ridge' down the diagonal). As λ decreases to 0, the problem reduces to the ordinary least squares, while as λ grows large, $\|\mathbf{c}\|$ becomes increasingly emphasized and minimization reduces it to zero. Suppose that the singular values of X are $\sigma_1, \ldots, \sigma_p$. What are the singular values of $(X'\,X \;+\; T'\,T)$? Also give the singular values of the *hat matrix* $H_\lambda = X\,(X'\,X + \lambda\,I)^{-1}\,X'$.

(c) Using the auto dataset in `auto_ridge.m`, let X be the matrix with three variables/columns *[Weight, Horsepower, Displacement]*, and let $\mathbf{y}$ be the vector *1 ./ MPG*. Also normalize X and $\mathbf{y}$ (i.e., replace them by their z-scores), so that all columns have the same scale. Plot the value of the 3 ridge regression coefficients $\mathbf{c}$ for at least 100 values of λ from 0 to $2\,\|X'\,X\|$. As λ increases, do the coefficients of $\mathbf{c}$ increase or decrease?

(d) Define $\mathrm{df}(\lambda) = \mathrm{trace}(\,H_\lambda\,)$, where H_λ is the hat matrix. What is df(0)?
Give a formula for $\mathrm{df}(\lambda)$ in terms of λ and the singular values of X.

(e) An 'optimal' value of λ will yield coefficients $\mathbf{c}$ that balance the RSS $\|\mathbf{y} - \widehat{\mathbf{y}}\|^2$ (notice that $\widehat{\mathbf{y}} = H_\lambda\,\mathbf{y}$ depends on λ) and the regularization error $\|\mathbf{c}\|^2$. Define the *Generalized Cross-Validation* (GCV) measure

$$GCV \;=\; \frac{n}{(n - \mathrm{df}(\lambda))^2}\;\|\,\mathbf{y} \;-\; \widehat{\mathbf{y}}\,\|^2.$$

Show how to find a value of λ that minimizes *GCV*. Ridge Regression is often implemented in this way.

4. **(Pseudo)Randomness (30 points)**

The *Henyey-Greenstein* distribution is often used to model the angle at which photons scatter. The photons might be passing through human tissue in a medical application or through the atmosphere in a climate modeling problem. Let $x = \cos(\theta)$. Then, the PDF (probability density function) of x is:

$$f(x; g) = \frac{1}{2} \frac{1 - g^2}{(1 + g^2 - 2gx)^{3/2}}, \qquad -1 \leqslant x \leqslant +1$$

for a parameter $g \in (-1, +1)$. In applications g is typically in the range from 0.7 to almost 1.

(a) Plot $f(x; g)$ versus x for $g = 0,\ 0.25,\ 0.5,\ 0.75$ and 0.97 with different colors, in the same figure.

(b) Use the *Monte Carlo* method to approximate the area under $f(x; g)$ for $g = 0.25$. Report your approximations for 5,000, 10,000, and 20,000 points. Plot your results in three separate figures and use different colors to identify *hits* and *misses*.

(c) Compute *analytically* the area under $f(x; g)$ for $-1 \leqslant x \leqslant +1$. Recall that $area = \int_{-1}^{+1} f(x; g)\ \mathrm{d}x$. Please provide your derivation. What is the absolute error of your three previous Monte Carlo approximations?

(d) Derive the mathematical expression for the ICDF (inverse cumulative density function) of $f(x; g)$. (*Hint:* Solve for y in the expression $\Pr(x \leqslant y) = \int_{-1}^{y} f(t; g)\ \mathrm{d}t$, where $\Pr(x \leqslant y)$ is a given constant between 0 and 1).

(e) Write a Matlab function `y = henyey( g )` which generates a random sample `y` that follows the *Henyey-Greenstein* distribution. Use the ICDF from previous answer — `rand(1)` will provide the "random" percent $\Pr(x \leqslant y)$ constant value that you need (i.e. the cumulative area under $f(x; g)$ for $x \leqslant y$).

(f) Use your `henyey()` function to generate 1,000 samples. Consider $g = 0.25$. Plot your samples in a histogram and superimpose the curve for $f(x; g)$.