Assignment 3

1. Suppose A is an $m \times n$ matrix with singular values

$$\sigma_{\max} \geq \cdots \geq \sigma_{\min} .$$

Suppose $\sigma_{\min} > 0$. Suppose that all norms are $2-norms$ (vector or matrix operator norms). Define $F(x) = Ax$. Consider the worst case condition number

$$\kappa(A) = \max_{x \neq 0} \|DF(x)\| \, \frac{\|x\|}{\|F(x)\|}$$

Show the following formula holds if $n > m$, if $n = m$, or if $n < m$

$$\kappa(A) = \frac{\sigma_{\max}}{\sigma_{\min}} .$$

2. Suppose A is an $m \times n$ matrix with singular values

$$\sigma_{\max} \geq \cdots \geq \sigma_{\min} .$$

Let ΔA be a small perturbation of A and let $\Delta\sigma_j$ be the estimated change in σ_j using first order perturbation theory. Show that these formulas are true

$$\sum_{jk} A_{jk}^2 = \sum_j \sigma_j^2$$
$$\sum_{jk} (\Delta A_{jk})^2 = \sum_j (\Delta\sigma_j)^2 .$$

What does this tell you about the sensitivity of singular values to small changes in A?

3. (*properties of convex functions*) You may assume the following basic fact from calculus for functions of one variable: If $u(t)$ is twice differentiable on the interval $[0, 1]$ and $u''(t) \geq 0$ for $0 \leq t \leq 1$, then $u(t) \leq 0$ for $0 \leq t \leq 1$.

   (a) Show that if $g''(t) \geq 0$ on an interval $[a, b]$ with $b > a$, then g is convex on that interval. This means that if $0 \leq \lambda \leq 1$ and $z(\lambda) = \lambda x + (1 - \lambda)y$ then

$$g(z(\lambda)) \leq \lambda g(x) + (1 - \lambda)g(y) .$$

   Hint. Define $u(\lambda) = \lambda g(x) + (1 - \lambda)g(y) - g(z(\lambda))$ and use the basic calculus fact.

(b) Suppose $f(x)$ is defined for $x \in \mathbb{R}^n$ and the hessian of f is positive semi-definite for all x. Show that f is convex. *Hint.* Use the trick of part (a). The fact that H is positive semi-definite tells you something about the second derivative of u with respect to λ.

4. (This exercise gives an example of an unstable algorithm for a well conditioned problem. The algorithm is unstable because it relies on a subproblem that is ill conditioned.)

   The problem comes from computing probabilities related to a simple hopping process. A *hopping process* is a random process in which a particle "hops" between neighboring "sites" at random times. A simple one dimensional hopping process "lives" on sites $\{0, 1, \cdots .n-1\}$ (the integers between 0 and $n-1$, including 0 and $n-1$. The location at time t is $X(t)$, which is a random site. The value of $X(t)$ is one of the integers $0, 1, \cdots, n-1$. We say X *hops* at time t if the value changes at that time. As a mathematical function, $X(t)$ is "piecewise constant", with discontinuities at the time with it hops.

   Suppose $X(t) = k$ with $0 \leq k \leq n-1$. In a time interval from t to $t+dt$, if $k < n-1$, it hops up to $k+1$ with probability $r_u\, dt$. If $k > 0$, the particle hops down to $k-1$ with probability $r_d\, dt$. If $X(t) = n-1$, then it cannot hop up, and if $X(t) = 0$ then it cannot hop down. The *occupation probabilities* are $p_k(t) = \Pr(X(t) = k)$. There is a small probability of having a hop in a small interval of time, but if you neglected it then there would be no hops at all. The probability of more than one hop is even smaller and (take my word for it) may be neglected.

   These probabilities satisfy a system of differential equations derived as follows. We denote conditional probability using the symbol "|", so $\Pr(A \mid B)$ is the probability of A conditional on B. Conditional probability allows to express $p_k(t+dt)$ in terms of $p_k(t)$, $p_{k-1}(t)$, and $p_{k+1}(t)$. If $X(t+dt) = k$, then $X(t) = k$ (most likely), or $X(t = k-1)$ and there was a hop up, or $X(t) = k+1$ and there was a hop down. The derivation neglects the possibility of more than one hop in interval dt. Here is the calculation, which some explanations after:

$$
\begin{aligned}
\Pr(X(t+dt) = k) = \;& \Pr(X(t+dt) = k \mid X(t) = k-1) \cdot \Pr(X(t) = k-1) \\
&+ \Pr(X(t+dt) = k \mid X(t) = k+1) \cdot \Pr(X(t) = k+1) \\
&+ \Pr(X(t+dt) = k \mid X(t) = k) \quad \cdot \Pr(X(t) = k) \\
p_k(t+dt) = \;& \Pr(\text{hop up}) \quad \cdot p_{k-1}(t) \\
&+ \Pr(\text{hop down}) \cdot p_{k+1}(t) \\
&+ \Pr(\text{no hop}) \quad \cdot p_k(t) \\
p_k(t+dt) &= r_u\, dt\, p_{k-1}(t) + r_d\, dt\, p_{k+1}(t) + (1 - r_u\, dt - r_d\, dt)\, p_k(t) \\
p_k(t+dt) &= p_k(t) + [r_u p_{k-1}(t) + r_d p_{k+1}(t) - (r_u + r_d) p_k(t)]\; dt\;.
\end{aligned}
$$

   The basic rule of conditional probability (if you haven't taken a big probability course) is that if A is an "event" ($X(t+dt) = k$ in this case) and

B, C, and D are distinct ways A can happen (B is $X(t) = k-1$, C is $X(t) = k$, etc.) then

$$\Pr(A) = \Pr(A \mid B) \cdot \Pr(B) + \Pr(A \mid C) \cdot \Pr(C) + \cdots \ .$$

The first equality in the derivation is this conditional probability formula. The probability of $X = k$ at $t+dt$ is the sum of the conditional probabilities multiplying the probabilities for the possible values of X at time t. The second equality says the same thing, using the above terminology and notation. The third inequality comes from substituting in the hopping probabilities. The probability of "no hop" is 1 minus the probability of a hop, which is $1 - r_u\,dt - r_d\,dt$. The notation is r_u for the rate to jump *up* and r_d for the rate to jump *down*. The code `MatrixExponential.py` uses $r_l = r_u + r_d$ for the "loss rate", which is the rate to jump out of site k. The corresponding probability to jump out of site k is $r_l\,dt$. The probability not to jump out is $1 - r_l\,dt$.

These formulas have to be modified if $k = 0$ (no down hops) or $k = n-1$ (no up hops). The modified formulas are

$$\begin{aligned} p_0(t+dt) &= \ p_0(t) \ + [r_d\,p_1(t) - r_u\,p_0(t)]\,dt \\ p_{n-1}(t+dt) &= p_{n-1}(t) + [r_u\,p_{n-2}(t) - r_d\,p_{n-1}(t)]\,dt \ . \end{aligned}$$

These relations may re-arranged and expressed in traditional calculus notation as

$$\begin{aligned} \frac{d}{dt}p_0(t) &= \qquad\qquad - \quad p_0(t)\,r_u \qquad + p_1(t)\,r_d \\ \frac{d}{dt}p_k(t) &= p_{k-1}(t)\,r_u r_u - p_k(t)(r_d + r_u) + p_{k+1}(t)\,r_d \ , \quad \text{for } 1 \le k \le n-2 \\ \frac{d}{dt}p_{n-1}(t) &= \ p_{n-2}(t)\,r_u \ - r_d\,p_{n-1}(t) \end{aligned}$$

This system if differential equations is expressed in matrix/vector form, by tradition, using a row vector (not column vector) for the probabilities $p(t) = (p_0(1), \cdots, p_{n-1}(t))$. The matrix form is the differential equations is

$$\frac{d}{dt}(p_0(1), \cdots, p_{n-1}(t)) = (p_0(1), \cdots, p_{n-1}(t)) \begin{pmatrix} -r_u & r_d & 0 & \cdots & 0 \\ r_u & -(r_u + r_d) & r_d & & \vdots \\ 0 & r_u & \ddots & \ddots & \\ \vdots & & \ddots & & r_d \\ 0 & \cdots & & r_u & -r_d \end{pmatrix}$$

In matrix/vector form, this is

$$\frac{d}{dt}p(t) = p(t)\,L \ .$$

The matrix L is the *generator* of the random hopping process. You can see that it is *tri-diagonal*, with non-zero elements only on the "main diagonal" and the nearest "off diagonals".

(a) A *diagonal scaling* (more properly, diagonal *re*-scaling) is

$$\widetilde{L} = W^{-1}LW \ , \ \ W = \mathrm{diag}(1, w_2, \cdots, w_{n-1}) \ .$$

Show that if W is non-singular, then the eigenvalues of L and $\widetilde{L}$ are the same. Find W so that $\widetilde{L}$ is symmetric. Conclude that the eigenvalues of L are real. Show that right eigenvectors of L are not left eigenvectors. *Hint for the last.* If $Lv = \lambda v$, then $WW^{-1}LWW^{-1}v = \lambda v$ so $\widetilde{L}\widetilde{v} = \lambda\widetilde{v}$, with suitable $\widetilde{v}$. [Not to hand in: any *sign symmetric* tridiagonal matrix (you supply the definition, allow for zeros on the off diagonal if you want) is similar to a symmetric tridiagonal matrix in this way. In differential equations, a *Sturm Liouville* operator (second order differential operator in one variable) is similar to a self-adjoint differential operator, using a diagonal "weighting function". Tri-diagonal matrices may be thought of as a discrete analogue of one-variable second order differential operators.]

(b) The code `MatrixExponential.py` implements three methods for solving the matrix differential equations $\frac{d}{dt}p = pL$ using the fundamental solution and matrix exponential. See `MatrixExponential.pdf` for more on this and a description of the methods. Experiment with the code on a variety of problems (change the dimension, the final time, how different the hopping rates are) to get a feel for which methods give accurate results for which problems. Look for problems that are not extreme that make the eigenvalue method look bad, and problems that make the matrix exponential method look bad. Note that you don't change the problem (or the solution algorithms) if you double the hopping rates and cut the final time in half.

(c) Modify the function `mee(L,t)` to return a tuple (Python term) consisting of the computed matrix exponential and the condition number of the eigenvector matrix R. Modify the function `meT(L,t,n)` to return its computed exponential and the largest norm $\left\| \frac{t^k}{k!}L^k \right\|$. Modify the output part of the main program (lines above 80) to add this information to the printout table. Comment on why/how well/not well this information explains the accuracy/inaccuracy of each method.

(d) (Not for credit, only if it seems interesting to you). Write a module that uses eigenvalyes/eigenvectors of the "symmetrized" matrix $\widetilde{L}$ to compute the exponential. This should be stable (unlike the unstable `mee`) because the symmetric eigenvalue/eigenvector problem is well conditioned. You might get the idea that using the symmetrized matrix is a cure-all. It isn't because most matrices cannot be symmetrized.