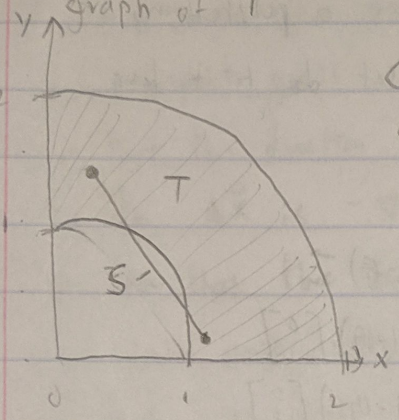
Vexing Convexity

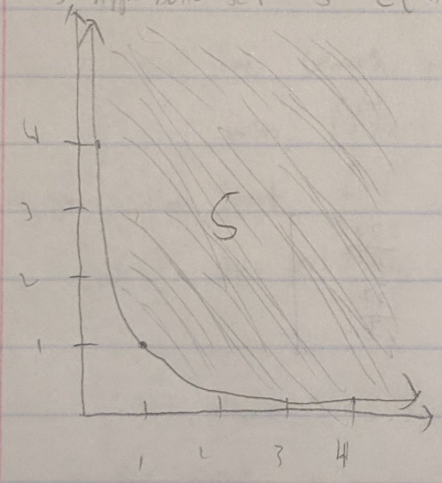
1) a) prove the following sets are convex or not

i) bent rectangle $T = \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 \geq 0, x_2 \geq 0, 1 \leq x_1^2 + x_2^2 \leq 4\}$



By inspection of the graph of set T, we can draw a line segment S with endpoints $\in T$, but S contains points $\notin T$, thus it is not convex

ii) hyperbolic set $S = \{(x_1, x_2) \in \mathbb{R}^2, x_1 > 0, x_2 > 0 \mid x_1 x_2 \geq 1\}$



S is convex if for any $x_1, x_2 \in S$ and any θ with $0 \leq \theta \leq 1$, where S is the set, then $\theta x_1 + (1-\theta) x_2 \in S$. Consider the case when $x, y \in S$ and $x_1, x_2 > y_1 y_2$. We can write x and y as the convex combination of two points $C = \theta x + (1-\theta) y$. Since $x > y$, $C > y$ which means $C_1 > y_1 \geq 1 \in S$ and $C_2 > y_2 \geq 1 \in S$, thus S is convex. Consider the case where $x, y \in S$ and $x_1 \leq y_1$, $x_2 \geq y_2$, thus $(y_1 - x_1)(y_2 - x_2) < 0$. We can write this as a convex combination of x_1, x_2, y_1, y_2 such that $C = (\theta x_1 + (1-\theta) y_1)(\theta x_2 + (1-\theta) y_2)$

FAIL

$$\Rightarrow \theta^2 x_1 x_2 + (1-\theta)^2 y_1 y_2 + (1-\theta)\theta x_1 y_2 + (1-\theta)\theta x_2 y_1$$

$$\Rightarrow \theta^2 (x_1 - y_1)(x_2 - y_2) + \theta(x_1 y_2 + x_2 y_1) - 2\theta y_1 y_2 + y_1 y_2$$

$$\Rightarrow \underbrace{\theta x_1 x_2 + (1-\theta) y_1 y_2}_{\geq 0} - \theta(1-\theta)\underbrace{(y_1 - x_1)(y_2 - x_2)}_{\text{negative}}$$

$$\Rightarrow \geq 1$$

thus given that $(y_1 - x_1)(y_2 - x_2) < 0$, the set S is convex

iii) Parallelogram $U = \left\{ \begin{bmatrix} x \\ y \\ z \end{bmatrix} = a\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + b\begin{bmatrix} 3 \\ 5 \\ 2 \end{bmatrix} + c\begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}, 0<a<1, 0<b<1, 0<c<1 \right\}$ (affine set)

Because the set U constructs a parallelogram that is a polyhedron, it is a convex set. It is convex also since it is an affine set due to it being a linear combination.

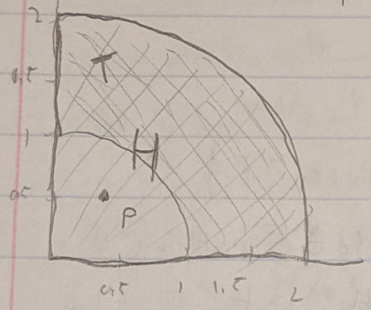
b) bent rectangle : Point in convex hull as convex combination

$P = (0.5, 0.5) \rightarrow \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \theta \vec{x}_1 + (1-\theta)\vec{x}_2$

$= \theta\begin{bmatrix} 1 \\ 0 \end{bmatrix} + (1-\theta)\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$= 0.5\begin{bmatrix} 1 \\ 0 \end{bmatrix} + (1-0.5)\begin{bmatrix} 0 \\ 1 \end{bmatrix}$

$= \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$



c) i) $f(x,y) = \frac{x^2}{y}$ $x \in \mathbb{R}$ $y > 0$

Second derivative test Hessian = $\begin{bmatrix} \frac{2}{y} & \frac{-2x}{y^2} \\ \frac{-2x}{y^2} & \frac{2x^2}{y^3} \end{bmatrix}$

$f_{xx} = \frac{2}{y}$ $f_{xy} = \frac{2x^2}{y^3}$ $f_{xy} = \frac{-2x}{y^2}$

$\det = \frac{4x^2}{y^4} - \frac{4x^2}{y^4} = 0 \geq 0$ this is convex

ii) $g(x) = 3x + e^{-x^2}$ $x \in \mathbb{R}$

$g_{xx} = -2e^{-x^2} + 4e^{-x^2}x^2$

$g(0)_{xx} = -2 \not\geq$ not convex

$g(1)_{xx} = \frac{2}{e} > 0$ not concave so is neither

iii) $h(x,y,z) = -x\log(x) - y^2 + 3z$ $x > 0$ $y \in \mathbb{R}$ $z \in \mathbb{R}$

(hessian calculator) $\rightarrow \begin{bmatrix} -\frac{1}{x} & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\det = 0 \geq 0$ thus it is convex

Gradient Descent Proofs

2) Consider an unconstrained min problem

$$\min f(\vec{x})$$

which is convex. Use gradient descent method on page 466

a) Suppose current guess is $\vec{x}_k$. How is the descent direction Δx chosen? Prove that this direction is feasible and improving at $\vec{x}_k$

- At $\vec{x}_k$, $\Delta\vec{x}$ is $-\nabla f(\vec{x}_k)$ and is the direction the algorithm steps in. We know that the direction is improving by definition if direction forms an acute angle with the negative gradient. Since $-\nabla f(\vec{x}_k)$ forms a 0° angle with itself, it is acute, thus improving.

b) Show exact line search problem is convex in t

$$\min f(\vec{x}_k + t\Delta\vec{x})$$

- Since we know that $f(\vec{x})$ is already convex, we can let $\vec{x}_k + t\Delta\vec{x}$ represent an arbitrary $\vec{x}$, meaning that $f(\vec{x}_k + t\Delta x)$ is also convex

c) Line Search vs backtracking

Exact line Search

- No extra parameters
- Slower since it calculates exact step size each iteration which is an optimization problem
- exact step size requires fewer iterations

Backtracking

- has 2 parameters!
  - α - how far we are away from exact tangent (estimate)
  - β - decrease in step size if
- Fast since it estimates the step size
- more iterations due to estimating step size
- Sensitive to bad hyper-parameters

Nonlinear Optimization Practice

3 a) Min $f(x) = x^3 \log(x) - 3x \quad x > 0$

$\therefore f'(x) = 3x^2 \log(x) + x^2 - 3$

$f''(x) = 6x \log(x) + 5x$

Choice: Newtons method and golden search since $f(x)$ is 1 dimensional and has simple derivatives. Also wanted to compare search method to an iterative one

Newton: $x = 1.2976$, 5 iterations, time = 0.002246 | Golden search was almost 2x

Golden: $x = 1.2976$, 42 iterations, time = 0.001460 | faster but had about 8x the iterations

b) Max $g(x,y) = -\frac{x^2}{y} + x \log - y^2 \quad x < 0 \; y > 0$

$\nabla g = \langle -\frac{2x}{y} + \log(-x) + 1, \frac{x^2}{y^2} - 2y \rangle$

$g_{xx} = \frac{1}{x} - \frac{2}{y} \quad g_{yy} = -2 - \frac{2x^2}{y^3} \quad g_{xy} = \frac{2x}{y^2}$ → Hessian $\begin{bmatrix} \frac{1}{x} - \frac{2}{y} & \frac{2x}{y^2} \\ \frac{2x}{y^2} & -2 - \frac{2y^3}{y^3} \end{bmatrix}$

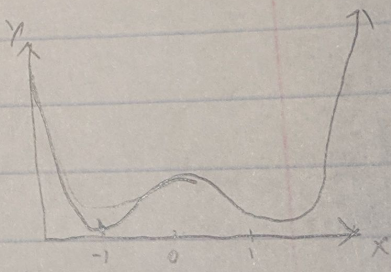
Choice: fminsearch since problem is multi dimensional and Newton since I calculated the Hessian

Newton: $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.1100 \\ -0.1422 \end{bmatrix}$, 6 iterations, time = 0.004204

fminsearch: $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.1100 \\ 0.1422 \end{bmatrix}$, 34 iterations, time = 0.002606

Newton converged faster but fminsearch was faster overall.

c) Min $h(x) = x^2(x^4 - 0.75x^3 + 1.5x^2 + 1.25x - 3) \quad x \in \mathbb{R}$ →



$h'(x) = 6x^5 - 3.75x^4 + 6x^3 + 3.75x^2 - 6x$

$h''(x) = 30x^4 - 15x^3 + 18x^2 + 7.5x - 6$

Choice: This function has two minima, but the first one is the global optimum around region 1, Chose to use exact 6D and fminsearch starting at -1

fminsearch: $x = -0.8296$, 17 iterations, time = 0.002614

exact: $x = 0.7185$, 2 iterations, time = 0.003815

→ We know from the graph that the exact method found the wrong minima. This is because it overshoted on its first iteration. Setting the starting point further back to -5 allows it to converge correctly w/ $x = -0.8295$ in 2 iterations and time 0.004516 seconds

a) $\min \; f(x,y,z) = (x^2 - 2)^2 + (\log(y) - 2)^2 + (z+y)^2 \quad x \in \mathbb{R} \quad y > 0 \quad z \in \mathbb{R}$

Choose: fminsearch and LINGO since the gradient of this function is hard to compute

fminsearch: $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1.4142 \\ 7.3890 \\ -7.3890 \end{bmatrix}$, 171 iterations, time = 0.003574

LINGO: $x = 1.41421$

$y = 7.38906$, 94 iterations, time = 0.39 seconds

$z = -7.38906$

b) LINGO calculates very slowly but converges faster

Retirement Planning

4) x_{1-6} stocks (AMZN, MSFT, BA, STF, bonds, BECU)

Covariance matrix Σ (6×6) is semi definite and invertable

x_i = % budget invested in asset i ; x_i is free

$\vec{x}$ = vector of budget percentages

$a = \sum_{i=1}^{6} x_i\mu_i = \mu^T\vec{x}$ is the expected annual return and $\sum_{i=1}^{6} x_i = 1$

where $\vec{\mu}$ is a vector of annual rates of return of each asset

Risk: $R = \vec{x}^T\Sigma\vec{x}$

want exactly 12% annual rate of return

a) Write as an optomization problem

$\min \ \vec{x}^T\Sigma\vec{x}$ minimize risk

$s.t. \ \mu^T\vec{x} = 0.12$ 12% return

$\sum_{i=1}^{6} x_i = 1$ invest all

$x_i \in \mathbb{R} \ \forall i$ x_i all reals

$$A = \begin{bmatrix} \mu_1 & 1 \\ \vdots & \vdots \\ \mu_6 & \end{bmatrix} \quad b = \begin{bmatrix} 0.12 \\ 1 \end{bmatrix}$$

b) Show problem is convex

we can prove the objective is convex or not by second derivative test

$f_0 = \vec{x}^T\vec{x}\,\Sigma \qquad f_0' = 2x\Sigma \qquad f_0'' = 2\Sigma$

our given Σ is positive, thus $f_0'' > 0$ and f_0 is convex.

(positive on the diagonal)

c) Used newtons method since problem has equality constraint.

resulting $\vec{x} = \begin{bmatrix} 0.1112 \\ 0.2189 \\ 0.1054 \\ 0.2556 \\ 0.2305 \\ 0.0784 \end{bmatrix}$ optimal risk = 0.3376

We want to invest 11.12% AMZN, 21.89% MSFT, 10.54% BA, 25.56% STF, 23.05% in US bonds, and 7.84% in a BECU account. We have a 33.76% risk of a lower than 12% return.

d) write lagrange dual and its optimization problem

lagrange = $\vec{x}^T \Sigma \vec{x} + \nu_1(\mu^T \vec{x} - 0.12) + \nu_2(\vec{1}^T \vec{x} - 1)$

- Set $\mathcal{L}' = (0)$

$\mathcal{L}' = 2\vec{x}\Sigma + \nu_1(\mu^T) + \nu_2(\vec{1}^T) = 0$

$= 2\vec{x}\Sigma = -\nu_1(\mu^T) - \nu_2(\vec{1}^T)$

$= \vec{x} = -\frac{1}{2}\Sigma^{-1}\nu_1(\mu^T) - \frac{1}{2}\Sigma^{-1}\nu_2(\vec{1}^T)$

substitute back into lagrange

$= \mathcal{L}D$

max $\mathcal{L}D$

s.t $\nu_1, \nu_2 \geq 0$

Solving the Processor Problem

5) - s is speed in range $S_t \in [S^{min}, S^{max}]$

- energy consumed is $\phi(S_t) = S_t^{\alpha}$ in period t

- total energy consumed is $E = \sum_{t=1}^{T} \phi(S_t)$

- Can run at different speeds in each T period

- there are n jobs

  - job i starts at time A and completes by D

  - job i requires $W_i \geq 0$ work to complete

- $\theta_{ti} \geq 0$ is fraction of processing power allocated to job i in period t

$$\sum_{i=1}^{n} \theta_{ti} = 1 \quad , \quad \sum_{t=A_i}^{D_i} \theta_{ti} S_t \geq W_i$$

- work completed on job i at time t is $S_{ti} = \theta_{ti} S_t$

Decision variables

a) We want to choose how to allocate power to each process at a given time period, which is S_{ti}. Since θ_{ti} is the amount of power allocated to job i at time t and S_t is processor speed at time t, the product $\theta_{ti} S_t$ is the matrix S_{ti} which is the amount of power allocated to each i jobs at time t, Therefore S_{ti} is the decision variable.

b) Optimization method

This is a problem where each entry row of t in S_{ti}, each i entry needs adjusted and updated on each iteration. Each row in t is its own optimization function w/ i variables. Each i entry in row t needs to converge to an optimal solution and thus can make use of gradient descent. So at an iteration of t, gradient descent can be used to solve each i given the current process requirements.

```
%problem 3
tol = 1e-9;

%a
f = @(x) ((x).^3).*log(x) - 3.*(x);
df = @(x) (3.*(x).^2).*log(x) + ((x).^2) -3;
dff = @(x) (6.*(x)).*log(x) + 5.*(x);

%newton
tic
[f_newt, iterf_newt] = newton(df,dff,1.5,tol);
toc

tic
[f_gold, iterf_gold] = golden_search(f,1,1.5,tol);
toc


%b
g = @(x) -1*( -(x(1) ^2) /x(2) + x(1) *log( -x(1) ) - x(2) ^2) ;
dg = @(x) [ -2*x(1)/x(2) + log(-x(1)) + 1;
    ( x(1).^2)/x(2).^2 - 2*x(2)];
dgg = @(x) [1/x(1) + -2/x(2)  , 2*x(1)/(x(2).^2) ;
    2*x(1)/(x(2).^2) , -2*(x (1).^2) /(x (2).^3) - 2];
%newton (cant start with [-1,1]?)
tic
[g_newt, iterg_newt] = newton2 (dg ,dgg ,[ -0.1;0.1] ,tol);
toc

%fminsearch
tic
[g_min,fval , exitflag , output ] = fminsearch (g ,[ -0.1;0.1]);
toc


%c
h = @(x) x.^6 - 0.75.*x.^5 + 1.5.*x.^4 + 1.25.*x.^3 - 3.*x.^2;
dh = @(x) 6.*x.^5 - 3.75.*x.^4 + 6.*x.^3 + 3.75.*x.^2 - 6.* x;
dhh = @(x) 30.*x.^4 - 15.*x.^3 + 18.*x.^2 + 7.5.*x -6;

tic
[h_l,fval , exitflag , output ] = fminsearch (h ,-1);
toc
```

```
%[h_back, opt, iterh_back] = grad_descent_backtracking(h,dh,1,tol,0.5,0.8)
tic
[h_ex, opt, iterh_ex] = grad_descent_exact(h,dh,-5,tol);
toc

%d
l = @(x) (x(1) ^2 -2) ^2 + (log(x(2) ) -2) ^2 + (x(3) +x(2) ) ^2;

%fmin search
tic
[x_l,fval , exitflag , output ] = fminsearch (l ,[0;1;0])
toc
```

```
%p6
%declare objective function and derivatives
f = @(x) (x')*c*x;
df = @(x) 2*x*c;
dff = @(x) 2*c;

%covariance matrix
c = [106.991461 23.892217 -27.843578 -59.063559 6.484768 0.007298
23.892217 93.627323 -117.56474 -40.45979 0.502487 0.006454
-27.843578 -117.56474 226.216329 16.106375 6.320048 -0.006226
-59.063559 -40.45979 16.106375 60.962192 -6.718355 -0.006654
6.484768 0.502487 6.320048 -6.718355 1.234519 -0.0009
0.007298 0.006454 -0.006226 -0.006654 -0.0009 0.000013];

%construct A matrix
A = [0.2083, 0.1874, 0.2111, 0.1095, 0.0239,0.001
    1, 1, 1, 1, 1, 1];

%construct b matrix
b = [0.12; 1];

%initialize starting weight matrix
w = zeros(6,1);

%use newton for equality constrained problems
[x,opt ,iter ,nu] = newton_eq (f,df ,dff ,w ,A, b ,1e-9)
```