Tutorial 3 (part 2)

Install these packages first

```
Pkg.add("QuadGK")
```

```
Pkg.add("Roots")
```

```
Pkg.add("Optim")
```

```
Pkg.add("Expectations")
```

Topic 1: Integration and Expectations

```
Integration
```

$$\int_0^\infty e^{-\frac{1}{2}x^2}\,dx$$

Will return two things: value and s.d. for the value. The value is 1.0000000000032583.

```
begin
    using QuadGK
    f1(x) = exp(-x^2 / 2) / sqrt(2*pi)
    q = quadgk(f1, -Inf, Inf)

    md"Will return two things: value and s.d. for the value.
    The value is $(q[1])."
end
```

```
Expectations
```

$$E[X^2]$$

```
0.9999999999999984
```

```
begin
    using Expectations

    dist1_1 = Normal(0, 1)
    E1_1 = expectation(dist1_1)

    f1_1(x) = x^2
    exp_value1_1 = E1_1(x -> f1_1(x))
end
```

Comparison between integration and expectation

$$max(0, X - D)$$

```
80.91701244547619
```

```
begin
    dist1_2 = Normal(100, 10^2)
    E1_2 = expectation(dist1_2)

    D1_2 = 35
    f1_2(x) = max(0, x - D1_2)
    exp_value1_2 = E1_2(x -> f1_2(x))
end
```

```
(80.5372, 6.3756e-7)
```

```
begin
    f1_3(x) = f1_2(x) * pdf(dist1_2, x)
    quadgk(f1_3, D1_2, Inf)
end
```

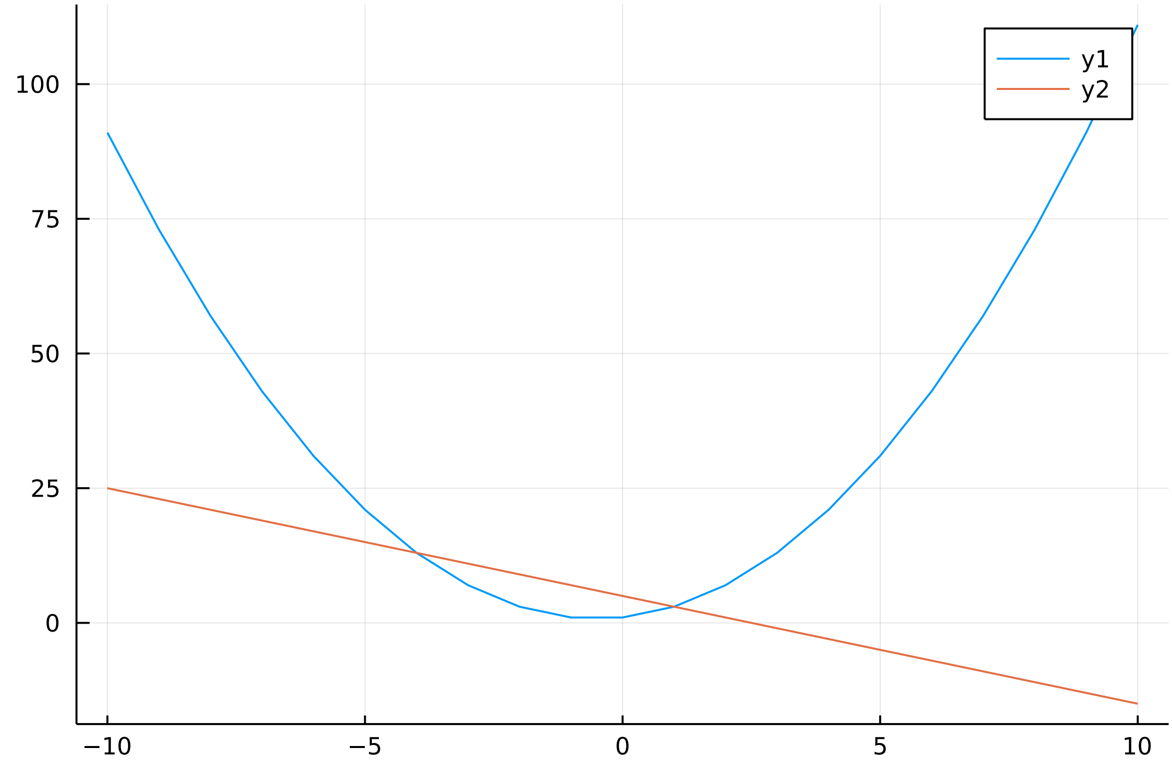
Topic 2: Finding the intersection of two function

```
[-4.0, 1.0]
```

```
begin
    using Roots

    g1(x) = x^2 + x + 1
    g2(x) = -2*x + 5

    delta(x) = g1(x) - g2(x)
    find_zeros(delta, -30, 30)
end
```




```
begin
    using Plots

    x_scale = (-10):10
    plot(x_scale, [g1,g2])
end
```

Topic 3: Find The Minimization Point for a Function

Rule:

(1) Define a function (topic3, here)

(2) Use "Optimize" function (function, lower bound, upper bound, algorithm)

```
Results of Optimization Algorithm
 * Algorithm: Brent's Method
 * Search Interval: [-20.000000, 20.000000]
 * Minimizer: -5.000000e-01
 * Minimum: 7.500000e-01
 * Iterations: 5
 * Convergence: max(|x - x_upper|, |x - x_lower|) <= 2*(1.5e-08*|x|+2.2e-16): true
 * Objective Function Calls: 6
```

```
begin
    using Optim

    topic3(x) = x^2 + x + 1

    opt = optimize(topic3, -20, 20, Brent())
end
```

optimal x = -0.5, optimal f = 0.75

Topic 4: Approximation

```
(1) Expected-Inventory-Level Approximation
```

$$Q = \sqrt{\frac{2\lambda(K + pn(r))}{h}}$$

$$r = F^{-1}(1 - \frac{Qh}{p\lambda})$$

$$g(r, Q) = h(r - \lambda L + \frac{Q}{2}) + \frac{K\lambda}{Q} + \frac{p\lambda n(r)}{Q}$$

$$E[(D-r)^+] = \int_r^\infty (d-r)f(d)dd = n(r)$$

```
EIL (generic function with 1 method)
```

```
begin
    function EIL(K, lambda, h, p, L, eplison, dist)
        theo_Q = sqrt(2 * K * lambda / h)

        pre_Q = 0
        new_Q = theo_Q
        pre_r = 0
        new_r = quantile(dist, 1 - theo_Q*h / (p*lambda))
        n_r = 0

        # when the difference between previous Q and new Q or the difference
        # between previous r and new Q is not larger than eplison, the loop should
        # stop and return the best_Q and best_r
        while((abs(new_Q - pre_Q) >= eplison) & (abs(new_r - pre_r) >= eplison))
            pre_Q = new_Q
            pre_r = new_r

            f(d) = (d - new_r) * pdf(dist, d)
            n_r = quadgk(f, new_r, Inf)[1]
            new_Q = sqrt(2 * lambda * (K + p*n_r) / h)

            new_r = quantile(dist, 1 - new_Q*h / (p*lambda) )
            if new_r < 0
                new_r = 0
            end
        end

        cost = h * (new_r - lambda*L + new_Q/2) + K * lambda / new_Q + p * lambda *
n_r / new_Q

        return new_Q, new_r, cost
    end
end
```

```
Distributions.Normal{Float64}(μ=108.33333333333333, σ=43.30127018922194)
```

```
begin
    using Distributions
```

```

    K1 = 8
    lambda1 = 1300
    h1 = 0.225
    p1 = 7.5
    L1 = 1/12
    eplison1 = 0.05
    mu1 = 1300/12
    sigma1 = 150/sqrt(12)
    dist1 = Normal(mu1, sigma1)
end
```

```
(318.555, 213.972, 95.4436)
```

```
Q1, r1, cost1 = EIL(K1, lambda1, h1, p1, L1, eplison1, dist1)
```

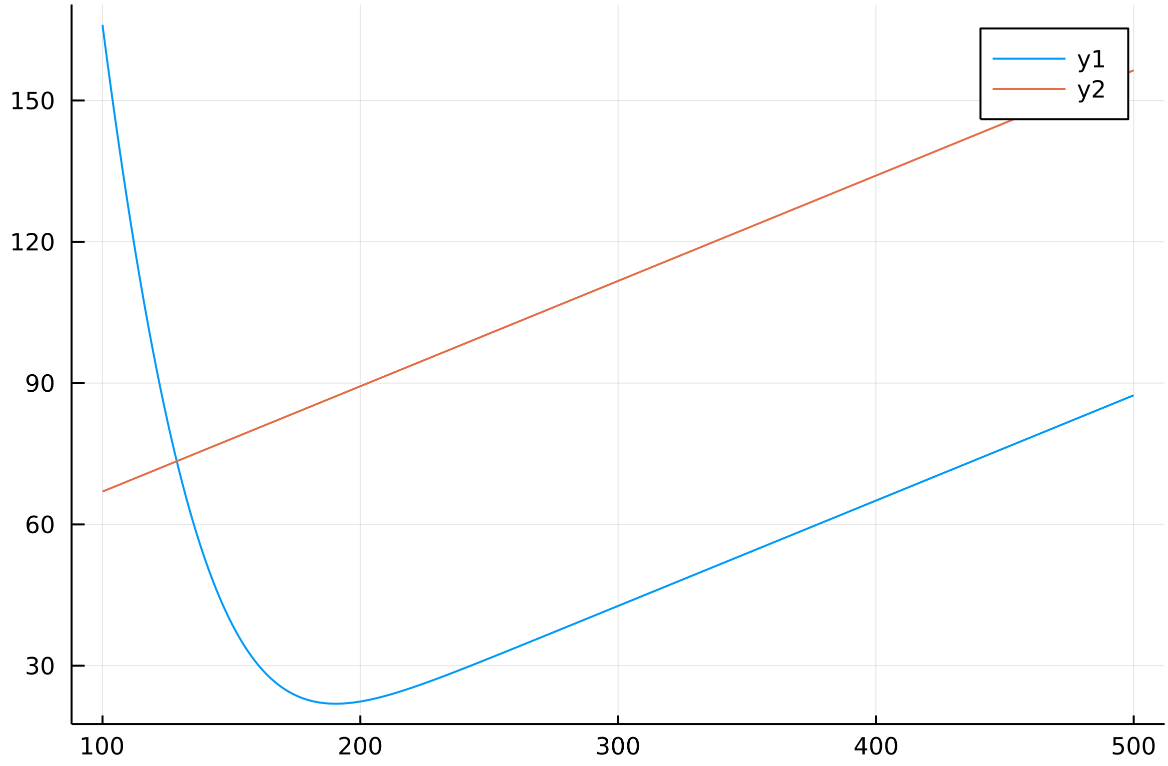
```
(2) EOQ with Backorder Approximation
```

$$Q^* = \sqrt{\frac{2K\lambda(h+p)}{hp}}$$

$$x^* = \frac{h}{h+p}$$

$$g(S) = h\int_0^S (S-d)f(d)dd + p\int_S^\infty (d-S)f(d)dd$$

To get the reorder point, we need to solve the equation $g(r) = g(Q+r)$




```
EOQB (generic function with 1 method)
```

```julia
begin
    function EOQB(K, lambda, h, p, dist)
        theo_Q = sqrt(2 * K * lambda * (h + p) / (h * p))

        function g(s)
            f1(x) = (s - x) * pdf(dist, x)
            f2(x) = (x - s) * pdf(dist, x)

            return h*quadgk(f1, 0, s)[1] + p*quadgk(f2, s, Inf)[1]
        end

        delta(r) = g(theo_Q + r) - g(r)

        r_star = find_zeros(delta, 100, 500)

        return theo_Q, r_star
    end
end
```

```
EOQB2 (generic function with 1 method)
```

```julia
begin
    function EOQB2(K, lambda, h, p, dist)
        theo_Q = sqrt(2 * K * lambda * (h + p) / (h * p))

        E = expectation(dist)
        function g(s)
            f1(x) = max(0, s - x)
            f2(x) = max(0, x - s)

            return h*E(x -> f1(x)) + p*E(x -> f2(x))
        end

        delta(r) = g(theo_Q + r) - g(r)

        r_star = find_zeros(delta, 100, 500)

        return theo_Q, r_star
    end
```

```
end
```

```
Distributions.Normal{Float64}(μ=108.33333333333333, σ=43.30127018922194)
```

```
begin
    K2 = 8
    lambda2 = 1300
    p2 = 7.5
    h2 = 0.225
    mu2 = 1300/12
    sigma2 = 150/sqrt(12)
    dist2 = Normal(mu2, sigma2)
end
```

```
(308.574, [128.812])
```

```
EOQB(K2, lambda2, h2, p2, dist2)
```

```
(308.574, [129.049])
```

```
EOQB2(K2, lambda2, h2, p2, dist2)
```

(3) EOQ+SS Approximation

For the parameter setting, check Example 5.5

$$Q = \sqrt{\frac{2K\lambda}{h}}$$

$$r = \mu + z_\alpha \sigma$$

$$\alpha = \frac{p}{p+h}$$

```
EOQ_SS (generic function with 1 method)
```

```
function EOQ_SS(K, lambda, h, p, mu, sigma)
    theo_Q = sqrt(2 * K * lambda / h)

    alpha = p / (p + h)
    r = mu + sigma * quantile(Normal(0, 1), alpha)

    function g(s)
        dist = Normal(mu, sigma)
        f1(x) = (s - x) * pdf(dist, x)
        f2(x) = (x - s) * pdf(dist, x)

        return h*quadgk(f1, 0, s)[1] + p*quadgk(f2, s, Inf)[1]
    end

    return theo_Q, r
end
```

```
43.30127018922194
```

```
begin
    K3 = 8
    lambda3 = 1300
    p3 = 7.5
    h3 = 0.225
    mu3 = 1300/12
    sigma3 = 150/sqrt(12)
end
```

```
(304.047, 190.337)
```

```
EOQ_SS(K3, lambda3, h3, p3, mu3, sigma3)
```

```
(4) Loss Function Approximation
```

For the parameter setting, check Example 5.6

$$g(r,Q) = \frac{K\lambda}{Q} + h(\frac{Q}{2} + r - \lambda L) + (h+p)B(r,Q)$$

$$B(r,Q) = \frac{1}{Q}[n^{(2)}(r) - n^{(2)}(r+Q)]$$

$$Q = \sqrt{\frac{2[K\lambda + (h+p)n^{(2)}(r)]}{h}}$$

$$n^{(2)}(x) = \frac{1}{2}E[([X-x]^+)^2] = \int_x^\infty n(y)dy$$

$$n(r) = \int_r^\infty (d-r)f(d)dd$$

And to solve r, you need to solve the equation $n(r) = \dfrac{hQ}{h+p}$

```
Loss_Approximation (generic function with 1 method)
```

```
function Loss_Approximation(K, lambda, h, p, L, dist, eplison)
    theo_Q = sqrt(2 * K * lambda / h)

    function n(r)
        f(d) = (d - r) * pdf(dist, d)
        return quadgk(f, r, Inf)[1]
    end

    # to solve r
    pre_r = 0
    lambda_out(x) = n(x) - h*theo_Q/(h+p)
    new_r = find_zeros(lambda_out, 100, 500)[1]

    # to solve Q
    n_2 = quadgk(n, new_r, Inf)[1]
    pre_Q = 0
```

```
    new_Q = sqrt(2 * (K * lambda + (h + p) * n_2) / h)
    while((abs(new_Q - pre_Q) >= eplison) & (abs(new_r - pre_r) >= eplison))
        pre_Q = new_Q
        pre_r = new_r

        lambda_inside(x) = n(x) - h*new_Q/(h+p)
        new_r = find_zeros(lambda_inside, 100, 500)[1]

        n_2 = quadgk(n, new_r, Inf)[1]
        new_Q = sqrt(2 * (K * lambda + (h + p) * n_2) / h)
    end

    #cost = K*lambda/new_Q + h*(new_Q/2 + new_r - lambda*L) + (h+p)/new_Q*n_2

    return new_Q, new_r
end
```

```
Distributions.Normal{Float64}(μ=108.33333333333333, σ=43.30127018922194)
```

```
begin
    K4 = 8
    lambda4 = 1300
    h4 = 0.225
    p4 = 7.5
    L4 = 1/12
    eplison4 = 0.05
    mu4 = 1300/12
    sigma4 = 150/sqrt(12)
    dist4 = Normal(mu4, sigma4)
end
```

```
(328.448, 126.868)
```

```
Loss_Approximation(K4, lambda4, h4, p4, L4, dist4, eplison4)
```

Algorithm 5.2

For detailed algorithm, check it on p.172 of your textbook

$$g(r, Q) = \frac{K\lambda}{Q} + h(\frac{Q}{2} + r - \lambda L) + (h+p)B(r, Q)$$

$$B(r, Q) = \frac{1}{Q}\int_r^{r+Q} E[(D-y)^+]dy$$

$$H(Q) = g(r(Q)) = g(r(Q) + Q)$$

$$A(Q) = QH(Q) - \int_0^Q H(y)dy$$

$$Q_d^* = \sqrt{\frac{2K\lambda(h+p)}{hp}}$$

$$H_0(Q) = H(Q) - g(S^*)$$

$$g(y) = hE[(y-D)^+] + pE[(D-y)^+]$$

and Q_0 is the Q that satisfies $QH_0(Q) = 2K\lambda$, S^* is the minimizer of $g(y)$

```
algorithm5_2 (generic function with 1 method)
```

```
begin
    function algorithm5_2(K, lambda, h, p, dist, eplison)
        Q_l = sqrt(2 * K * lambda * (h + p) / (h * p))

        function g(s)
            f1(x) = (s - x) * pdf(dist, x)
            f2(x) = (x - s) * pdf(dist, x)

            return h*quadgk(f1, 0, s)[1] + p*quadgk(f2, s, Inf)[1]
        end

        function H(q0)
            diff(r) = g(r) - g(r + q0)
            r_value = find_zeros(diff, -500, 500)[1]
            return g(r_value)
        end

        g_star = optimize(g, -500, 500, Brent()).minimum
        H_0(q1) = H(q1) - g_star
        H_eq(q2) = q2 * H_0(q2) - 2 * K * lambda
        Q_u = find_zeros(H_eq, 0.1, 800)[1]

        A = typemax(Int64)
        Q = typemax(Int64)
        r = typemax(Int64)
        while(abs(A - K*lambda) > eplison)
            Q = (Q_u + Q_l) / 2

            diff2(x) = g(x) - g(Q + x)
            r = find_zeros( diff2, -500, 500)[1]

            A = Q * H(Q) - quadgk(H, 0.001, Q)[1]
            if A > K * lambda
                Q_u = Q
            else
                Q_l = Q
            end
        end

        return Q, r
    end
end
```

```
Distributions.Normal{Float64}(μ=108.33333333333333, σ=43.30127018922194)
```

```
begin
    K5 = 8
    lambda5 = 1300
    h5 = 0.225
    p5 = 7.5
    eplison5 = 2
    mu5 = 1300/12
    sigma5 = 150/sqrt(12)
    dist5 = Normal(mu5, sigma5)
end
```

```
(329.384, 126.963)
```

```
algorithm5_2(K5, lambda5, h5, p5, dist5, eplison5)
```

Algorithm 5.3

For detailed algorithm, check it on p.179 of your textbook

And the for the parameter setting you can check Example 5.8

$$g(r, Q) = \frac{K\lambda + \sum_{y=r+1}^{r+Q} g(y)}{Q}$$

$$g(y) = hE[(y-D)^{+}] + pE[(D-y)^{+}]$$

where S^* minimizes $g(y)$

```
algorithm5_3 (generic function with 1 method)
```

```
function algorithm5_3(K, lambda, h, p, dist)
    Q = 1
    S_star = 0
    while(cdf(dist, S_star) < (p/(p+h)))
        S_star = S_star + 1
    end
    r = S_star - 1

    function g(x)
        appro_bound = 10

        n = h * sum((x .- [j for j in 0:x]) .*
        pdf.(dist, [j for j in 0:x]))

        n_bar = p * sum(([j for j in x:(appro_bound*lambda)] .- x) .*
        pdf.(dist, [j for j in x:(appro_bound*lambda)]))

        return n + n_bar
    end

    function f(r, Q)
        sum = 0
        for i in (r+1):(r+Q)
            sum = sum + g(i)
        end

        return sum
    end

    g_pre = K*lambda/Q + f(r, Q)/Q
    r_pre = r
    while(true)
        if g(r_pre) < g(r_pre + Q + 1)
            r_new = r_pre - 1
        else
            r_new = r_pre
        end

        g_new = (f(r_new, Q+1) + K*lambda)/(Q+1)
        if g_new > g_pre
            break
        else
            Q = Q + 1
        end

```

```
            g_pre = g_new
            r_pre = r_new
        end


        return r_pre, Q, g_pre
    end
```

```
Distributions.Poisson{Float64}(λ=3.0)
```

```
begin
    K6 = 100
    lambda6 = 1.5
    h6 = 20
    p6 = 150
    L6 = 2
    dist6 = Poisson(lambda6*L6)
end
```

```
(3, 5, 107.923)
```

```
algorithm5_3(K6, lambda6, h6, p6, dist6)
```