MTH3045: Statistical Computing

Dr. Ben Youngman
b.youngman@exeter.ac.uk
Laver 817; ext. 2314

2/3/2026

Week 8 lecture 1

Simpson's rule

- The midpoint rule works simply by approximating $f(x)$ over a sub-interval of $[a, b]$ by a horizontal line
- The trapezium rule (which we'll overlook) assumes a straight line
- Simpson's rule is derived from a quadratic approximation and given by

$$\int_a^b f(x)\mathrm{d}x \simeq \frac{h}{3}\left(f(a) + 4\sum_{i=1}^{N/2} f(x_{2i}^*) + 2\sum_{i=1}^{N/2-1} f(x_{2i+1}^*) + f(b) \right),$$

  where $x_i^* = a + ih$ for $i = 1, \ldots, N-1$ and $h = (b-a)/N$
- Note that Simpson's rule requires one more evaluation of f than the midpoint rule
  - however, a benefit of that extra evaluation is that its error reduces to $O(h^4)$

Example: Simpson's rule I

- Now use `R` and Simpson's rule to approximate the integral $\int_0^1 \exp(x)\mathrm{d}x = \exp(1) - 1$ with $N = 10$, 100 and 1000, compare the relative absolute error for each, and against those of the midpoint rule in Example 4.5
- We already have `true` and `N_vals` from Example 4.5, and we can use a similar `for` loop to approximate the integral using Simpson's rule
- The main difference is that we do not multiply $f()$ when evaluated at the nodes by the same value: instead sometimes it is multiplied by 4 and other times it is multiplied by 2

Example: Simpson's rule II

- The integral approximations are stored as `simpson`

```
N_vals <- 10^c(1:3)
true <- exp(1) - 1
simpson <- numeric(length(N_vals))
for (i in 1:length(N_vals)) {
  N <- N_vals[i]
  h <- 1 / N
  nodes <- h * 1:(N - 1)
  simpson[i] <- exp(0)
  simpson[i] <- simpson[i] + 4 * sum(exp(nodes[2 * c(1:(N / 2)) - 1]))
  simpson[i] <- simpson[i] + 2 * sum(exp(nodes[2 * c(1:((N / 2) - 1))]))
  simpson[i] <- simpson[i] + exp(1)
  simpson[i] <- h * simpson[i] / 3
}
print(simpson, digits = 12)
```

```
## [1] 1.71828278192 1.71828182855 1.71828182846
```

- We print this to 11 decimal places so we can see where the approximations changes with N

Example: Simpson's rule III

- Finally we calculate the relative absolute errors, `rel_err_simp`,

```
rel_err_simp <- abs((true - simpson) / true)
rel_err_simp
```

```
## [1] 5.548949e-07 5.555503e-11 5.815115e-15
```

- We see a dramatic improvement in the accuracy of approximation that Simpson's rule brings, with relative absolute errors of the same order of magnitude as those form the midpoint rule using $N = 1000$ achieved with $N = 10$ for Simpson's rule
- Note, though, that for given N, Simpson's rule requires $N + 1$ more evaluations of $f()$

Challenges I

- Go to Challenges I of the week 8 lecture 1 challenges at https://byoungman.github.io/MTH3045/challenges

Gaussian quadrature I

- We've seen that Simpson's rule can considerably improve on the midpoint rule for approximating integrals
- However, we might still consider both restrictive in that they consider an equally-spaced set of nodes
- **Definition**: Consider $g(x)$, a polynomial of degree $2N-1$, and a fixed weight function $w(x)$
- Then, the **Gauss-Legendre quadrature rule** states that

$$\int_a^b w(x)g(x)\mathrm{d}x = \sum_{i=1}^{N} w_i g(x_i),$$

where, for $i = 1, \ldots, N$, w_i and x_i depend on $w(x)$, a and b, but not $g(x)$

Gaussian quadrature II

- The Gauss-Legendre quadrature rule is the motivation for **Gaussian quadrature**, whereby we assume that the integral we're interested in can be well-approximated by a polynomial
- This results in the approximation

$$\int_a^b f(x)\mathrm{d}x \simeq \sum_{i=1}^{N} w_i f(x_i)$$

  for a fixed set of x values, x_i with corresponding weights w_i, for $i = 1, \ldots, N$
- There are many rules for choosing the weights, w_i, but (perhaps fortunately) we won't go into them in detail in MTH3045
- Instead, we'll just consider the function `pracma::gaussLegendre()` (for which you'll need to install the `pracma` package), where `pracma::gaussLegendre(N, a, b)` produces `N` nodes and corresponding weights on the interval [a,b], with $N =$ `N`, $a =$ `a` and $b =$ `b`

Gaussian quadrature III

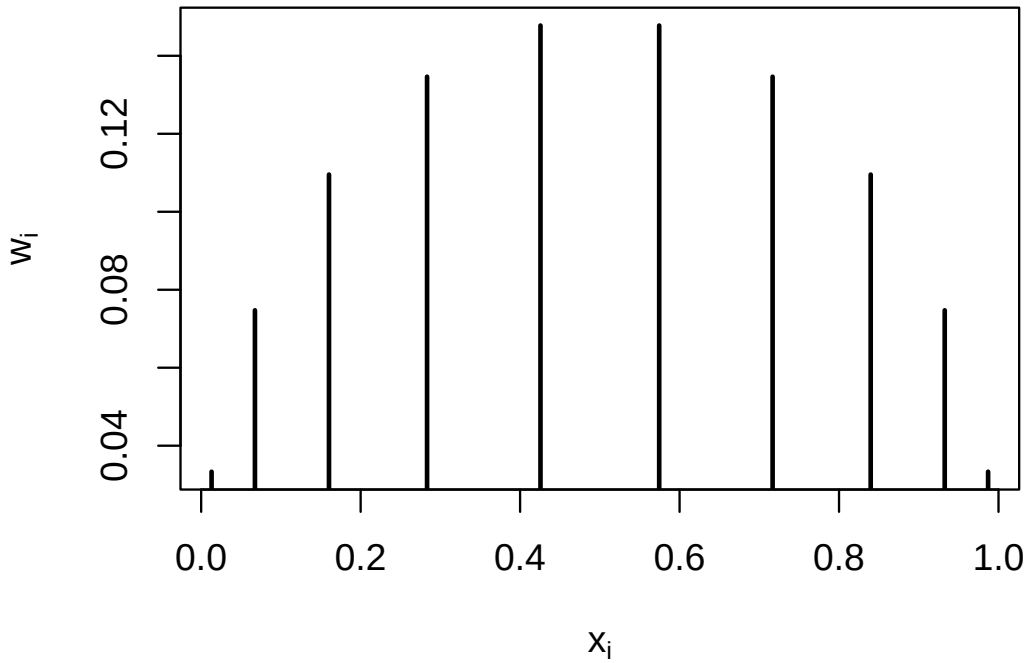
- The following produces nodes and weights for $N = 10$ on $[0, 1]$

```
gq <- pracma::gaussLegendre(10, 0, 1)
gq
```

```
## $x
##  [1] 0.01304674 0.06746832 0.16029522 0.28330230 0.42556283 0.57443717
##  [7] 0.71669770 0.83970478 0.93253168 0.98695326
##
## $w
##  [1] 0.03333567 0.07472567 0.10954318 0.13463336 0.14776211 0.14776211
##  [7] 0.13463336 0.10954318 0.07472567 0.03333567
```

Gaussian quadrature IV

- The plots below shows the nodes and their weights



- The nodes are spread further apart towards the middle of the $[0, 1]$ range, but given more weight
- Note that `pracma::gaussLegendre()` is named so because it implements Gauss-Legendre quadrature, i.e. Gaussian quadrature with Legendre polynomials[1]

[1] Wikipedia has a useful pages on Gaussian quadrature and on Legendre polynomials, should you want to read more on them

Example: Gaussian quadrature I

- Now use R and Gauss-Legendre quadrature to approximate the integral $\int_0^1 \exp(x)\mathrm{d}x$ with $N = 10$
- Explore what value of N gives a comparable estimate to that of the midpoint rule with $N = 100$ based on relative absolute error
- We can re-use `true` from Example 4.5 and then we'll consider $N = 10$, 4 and 3, which we'll call `N_vals`, and store the resulting integral approximations in `gauss`.

```
N_vals <- c(10, 4, 3)
gauss <- numeric(length(N_vals))
for (i in 1:length(N_vals)) {
  N <- N_vals[i]
  xw <- pracma::gaussLegendre(N, 0, 1)
  gauss[i] <- sum(xw$w * exp(xw$x))
}
gauss
```

```
## [1] 1.718282 1.718282 1.718281
```

Example: Gaussian quadrature II

- The relative absolute errors, `rel_err_gauss`,

```
true <- exp(1) - 1
rel_err_gauss <- abs((true - gauss) / true)
rel_err_gauss
```

```
## [1] 2.584496e-16 5.429651e-10 4.795992e-07
```

  show that, having considered $N = 3, 4, 10$, choosing $N = 3$ for Gaussian quadrature gives closest relative absolute error to that of the midpoint rule with $N = 100$, which really is quite impressive

- Note, though, that $f(x) = \exp(x)$ is a very smooth function
  - for wiggler functions, larger N is likely to be needed, and improvements in performance, such as Gaussian quadrature over the midpoint rule, might be significantly less

Challenges II

- Go to Challenges II of the week 8 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

Week 8 lecture 2

Example: Poisson marginal approximation using Gaussian quadrature I

- Consider a single random variable $Y \mid \lambda \sim \text{Poisson}(\lambda)$, where we can characterise our prior beliefs about λ as $\lambda \sim \text{N}(\mu, \sigma^2)$
- Use Gaussian quadrature with $N = 7$ to estimate the marginal pdf of Y if $\mu = 10$ and $\sigma = 3$
- The marginal pdf of Y is given by

$$f(y) = \int_{-\infty}^{\infty} f(y \mid \lambda) f(\lambda) \mathrm{d}\lambda$$

- *Remark*: The *three sigma rule* is a heuristic rule of thumb that 99.7% of values lie within three standard deviations of the mean

Example: Poisson marginal approximation using Gaussian quadrature II

- Hence for the $\mathrm{N}(10, 3^2)$ distribution we should expect 99.7% of values to lie within $10 \pm 3 \times 3$
- Hence we'll take this as our range for the Gaussian quadrature nodes.

```
mu <- 10
sigma <- 3
N <- 7  # no. of nodes
xw <- pracma::gaussLegendre(
        N,
        mu - 3 * sigma, # left-hand end
        mu + 3 * sigma # right-hand end
)
xw
```

```
## $x
## [1]  1.458029  3.326219  6.347394 10.000000 13.652606 16.673781 18.541971
##
## $w
## [1] 1.165365 2.517349 3.436470 3.761633 3.436470 2.517349 1.165365
```

which are stored as `xw$x` with corresponding weights `xw$w`, $w_1, \ldots, w_N$

Example: Poisson marginal approximation using Gaussian quadrature III

- Next we want a set of values at which to evaluate $f(y)$, and for this we'll choose $0, 1, \ldots, 30$, which we can create in R with

```
y_vals <- 0:30
```

- Then we can estimate $f(y)$ as

$$\hat{f}(y) \simeq \sum_{i=1}^{N} w_i f(y \mid \lambda_i^*) f(\lambda_i^*)$$

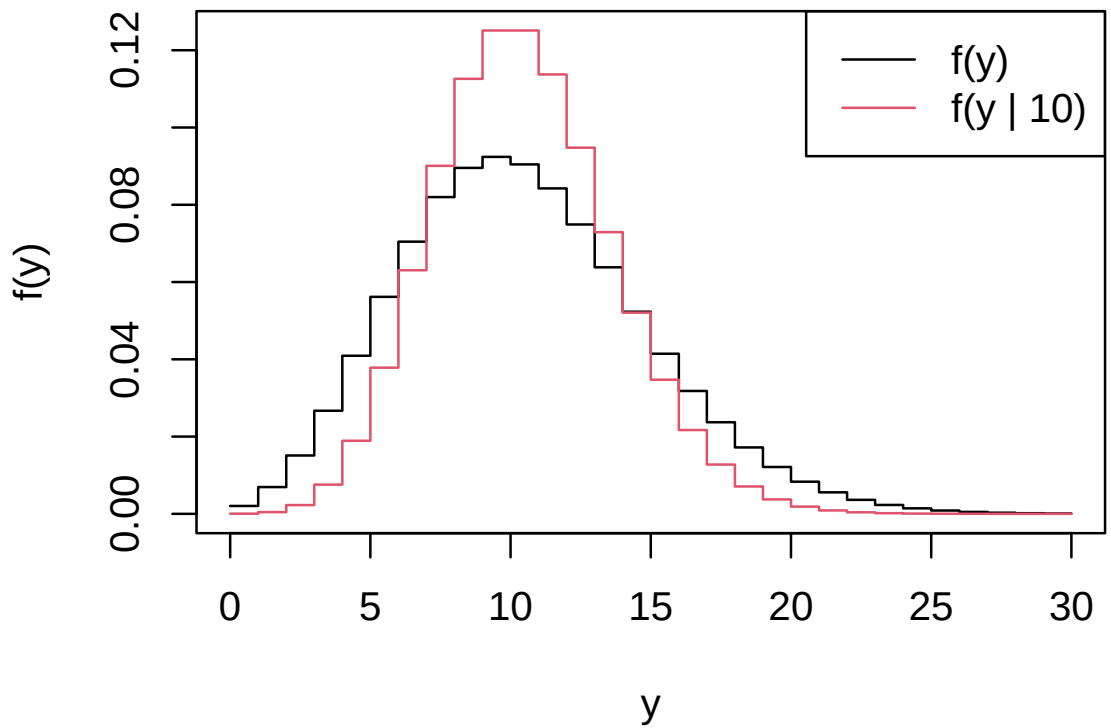
- The following code gives $\hat{f}(y)$ as `fhat` for y in the set of values `y_vals`

```
m <- length(y_vals)
fhat <- numeric(m)
for (i in 1:m) {
  fhat[i] <- sum(xw$w * dpois(y_vals[i], xw$x) *
                   dnorm(xw$x, mu, sigma))
}
```

Example: Poisson marginal approximation using Gaussian quadrature IV

- Finally, we'll plot $\hat{f}(y)$ against the pdf of the Poisson(10) distribution

```
matplot(y_vals, cbind(fhat, dpois(y_vals, mu)), lwd = 1,
        col = 1:2, lty = 1, type = 'l', xlab = 'y', ylab = 'f(y)')
legend('topright', c("f(y)", "f(y | 10)"),
       lty = 1, col = 1:2)
```



- $f(y)$ is broader than $f(y \mid 10)$, which is to be expected given that $f(y)$ integrates out the variability in λ given by the $\mathrm{N}(10, 3^2)$ distribution

One-dimensional numerical integration in R

- Unsurprisingly, R has a function for one-dimensional numerical integration
- It's called `integrate()`
- It uses a method that builds on Gaussian quadrature, but we won't go into its details
- Use of `integrate()`, however, is fairly straightforward

Example: Integration with `integrate()`

- Evaluate the integral $\int_0^1 \exp(x)dx = \exp(1) - 1$ using R's `integrate()` function with $N = 10$ and report its relative absolute error
- We can use the following code, where the first argument to `integrate()` is the function we're integrating, the second and third are the lower and upper ranges of the definite integral, and `subdivisions` is the maximum number of nodes to use in the approximation, which defaults to 100.

```
true <- exp(1) - 1
estimate <- integrate(function(x) exp(x), 0, 1, subdivisions = 10)
estimate
```

```
## 1.718282 with absolute error < 1.9e-14
```

```
rel_err <- abs((true - estimate$value) / true)
rel_err
```

```
## [1] 1.292248e-16
```

- Note above that the absolute error is similarly tiny to that of Gaussian quadrature above
- The values themselves, being so close to the machine tolerance, are incomparable
  - but we can be sure that the approximation is incredibly accurate

Multi-dimensional quadrature I

- Now suppose that we want to integrate a multi-variable function over some finite range
- For simplicity, we'll just consider the case of function of two variables first, e.g. $f(x, y)$
- Then, applying one-dimensional quadrature, we have that

$$\int f(x, y_j)\mathrm{d}x \simeq \sum_{i=1}^{M} w_{x,i} f(x_i, y_j)$$

and then, by another application of quadrature, we have that

$$\int \int f(x, y)\mathrm{d}x\mathrm{d}y \simeq \sum_{i=1}^{M} w_{x,i} \sum_{j=1}^{N} w_{y,j} f(x_i, y_j)$$

- Note that the weight sequences $w_{x,i}$, $i = 1, \ldots, M$, and $w_{y,j}$, $j = 1, \ldots, N$, can be specified separately, i.e. according to different quadrature rules

Multi-dimensional quadrature II

- More generally, we therefore have that

$$\int \dots \int f(x_1, \dots, x_d) \mathrm{d}x_1 \dots, \mathrm{d}x_d = \int f(\mathbf{x}) \mathrm{d}\mathbf{x}$$

$$\simeq \sum_{i_1=1}^{N_1} \dots \sum_{i_d=1}^{N_d} w_{x_1,i_1} \dots w_{x_d,i_d} f(x_{1,i_1}, \dots, x_{d,i_d})$$

- *Remark*: In practice, multi-dimensional quadrature is only feasible for small numbers of dimensions
- For example, consider a d-variable function f, such that each dimension has N nodes
- This will require N^d evaluations of f
- Some useful numbers to draw upon are 10^6, and $3^{15} \simeq 14348907$

Example: Two-variable numerical integration using the midpoint rule I

- Use the midpoint rule to approximate

$$I = \int_0^1 \int_0^1 \exp(x_1 + x_2) \mathrm{d}x_1 \mathrm{d}x_2$$

  with 10 nodes for each variable, and estimate its relative absolute error

- The following sets d and finds the integral's true value, `true`

```
d <- 2
true <- (exp(1) - 1)^d
```

- We want $N = 10$ nodes per variable

```
N <- 10
```

- Then we'll use the following nodes for x_1 and x_2 on $[0, 1]$

```
x1 <- x2 <- (1:N - .5) / N
```

Example: Two-variable numerical integration using the midpoint rule II

- The approximation to the integral is given by

$$\hat{I} = \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} f(x_{1i}, x_{2j})$$

  where $w_{ij} = N^{-d}$ for $i, j = 1, \ldots, N$

- This can can be evaluated in R with

```
midpoint <- 0
w <- 1/(N^d)
for (i in 1:N) for (j in 1:N)
  midpoint <- midpoint + w * exp(x1[i] + x2[j])
```

  where `midpoint` calculates $\hat{I}$ above

Example: Two-variable numerical integration using the midpoint rule III

- The following give the true integral, alongside its midpoint-rule based estimate, and the absolute relative error of the estimate, `rel.err`

```
c(true = true, midpoint = midpoint, rel.err = abs((midpoint - true) / true))
```

```
##          true     midpoint      rel.err
## 2.9524924420 2.9500332614 0.0008329168
```

- We see that the estimate is reasonably accurate, but have had to evaluate $f(x_1, x_2)$ 100 times

Challenges I

- Go to Challenges I of the week 8 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

Example: Two-variable numerical integration using Gaussian quadrature

- Use Gaussian quadrature to approximate I from Example 4.10 with 4 nodes for each variable, and estimate its relative absolute error
- We can take `d` and `true` from Example 4.10
- Then we calculate $\hat{I}$ as above, but using the integration nodes and weights of Gaussian quadrature.

```
N <- 4
xw1 <- xw2 <- pracma::gaussLegendre(N, 0, 1)
gq <- 0
for (i in 1:N) for (j in 1:N)
  gq <- gq + xw1$w[i] * xw2$w[j] * exp(xw1$x[i] + xw2$x[j])
gq
```

```
## [1] 2.952492
```

- Again Gaussian quadrature gives an incredibly accurate estimate

```
c(true = true, gauss_quad = gq, rel.err = abs((gq - true) / true))
```

```
##         true   gauss_quad      rel.err
## 2.952492e+00 2.952492e+00 1.085930e-09
```

yet now we've only had to evaluate $f(x_1, x_2)$ 16 times

Challenges II

- Go to Challenges II of the week 8 lecture 2 challenges at https://byoungman.github.io/MTH3045/challenges

Week 8 lecture 3

Example: Five-variable numerical integration using Gaussian quadrature I

- Use Gaussian quadrature to approximate

$$I = \int_0^1 \dots \int_0^1 \exp\left(\sum_{i=1}^{5} x_i\right) \mathrm{d}x_1 \dots \mathrm{d}x_5$$

  with 4 nodes for each variable, and estimate its relative absolute error.

- We can proceed as in Example 4.11 by storing d as `d` and I as `true`

```
d <- 5
true <- (exp(1) - 1)^d
```

- Now we'll need a new tactic, because forming `xw1`, `xw2`, ..., `xw5` would be rather laborious
- Having the same nodes and weights for each variable, i.e. for x_1, x_2, ..., x_5, simplifies the following code a bit

Example: Five-variable numerical integration using Gaussian quadrature II

- We'll start by setting N and getting the quadrature nodes and weights, which will be repeated for each variable

```
N <- 4
xw <- pracma::gaussLegendre(N, 0, 1)
```

- Then we want to put together all the combinations of the nodes, `X`, and weights, `W`, that will be used

```
xxww <- lapply(1:d, function(i) xw)
X <- expand.grid(lapply(xxww, '[[', 1))
W <- expand.grid(lapply(xxww, '[[', 2))
```

Example: Five-variable numerical integration using Gaussian quadrature III

- We then want multiply all the combinations of weights, for which we could use `apply(..., 1, prod)`, but instead we can use `rowSums()` in the following way

```
w <- exp(rowSums(log(W)))
```

- We then calculate $\hat{I}$ and store this as gq

```
gq <- sum(w * exp(rowSums(X)))
```

  and see that we still get an excellent approximation to I

```
c(true = true, gauss_quad = gq, rel.err = abs(gq - true) / true)
```

```
##         true   gauss_quad      rel.err
## 1.497863e+01 1.497863e+01 2.714825e-09
```

  but have now evaluated $\exp(\sum_{j=1}^{5} x_i)$ 1024 times

Challenges I

- Go to Challenges I of the week 8 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

Laplace's method

An aside on Taylor series in one dimension

- In MTH1002 you met Taylor series expansions
- The expansion is at the centre of many useful statistical approximations
- **Theorem**: Consider a function $f(x)$ in the region of a point x_0 that is infinitely differentiable
- Then

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \frac{f^{(3)}(x_0)}{3!}(x - x_0)^3 + \dots$$

  where $f'(x)$ and $f''(x)$ denote the first and second derivatives of $f(x)$ w.r.t. x, respectively, and $f^{(n)}(x)$ denotes the nth derivative, for $n = 3, 4, 5, \dots$

- From a statistical perspective we're often just looking to approximate $f(x)$ up to second-order terms, and hence we may work with the truncated expansion

$$f(x) \simeq f(x_0) + (x - x_0)f'(x_0) + \frac{1}{2}(x - x_0)^2 f''(x_0). \qquad (4.2)$$

Laplace's method: definition

- **Definition**: Consider the integral

$$I_n = \int_{-\infty}^{\infty} \mathrm{e}^{-nf(x)} \mathrm{d}x, \tag{4.3}$$

where $f(x)$ is a *convex* function with minimum at $x = \tilde{x}$, so $f'(\tilde{x}) = 0$

- The integral can be approximated, using **Laplace's method**, as

$$I_n \simeq \mathrm{e}^{-n[f(\tilde{x})]} \sqrt{\frac{2\pi}{n|f''(\tilde{x})|}}, \tag{4.4}$$

which is also sometimes referred to as the *Laplace approximation*[2]

[2] Pierre-Simon Laplace (23 March 1749 – 5 March 1827) was a "French mathematician, astronomer, and physicist who was best known for his investigations into the stability of the solar system. Laplace successfully accounted for all the observed deviations of the planets from their theoretical orbits by applying Sir Isaac Newton's theory of gravitation to the solar system, and he developed a conceptual view of evolutionary change in the structure of the solar system. He also demonstrated the usefulness of probability for interpreting scientific data", according to Britannica.com. He developed the Laplace transform, in addition to Laplace's method, and, with Abraham de Moivre, is responsible for the de Moivre–Laplace theorem, which approximates the binomial distribution with a normal distribution.

Laplace's method: derivation I

- Laplace's method can be derived from the truncated Taylor expansion of Equation (4.2)
- If we expand $f(x)$ about its global maximum, which we'll denote by $\tilde{x}$, then

$$f(x) \simeq f(\tilde{x}) + \frac{1}{2}(x - \tilde{x})^2 f''(\tilde{x})$$

  since $f'(\tilde{x}) = 0$
- Substituting this in we get

$$\begin{aligned} I_n &\simeq \int_{-\infty}^{\infty} \exp\left\{-n\left[f(\tilde{x}) + \frac{1}{2}(x - \tilde{x})^2 f''(\tilde{x})\right]\right\} \mathrm{d}x \\ &= \mathrm{e}^{-n[f(\tilde{x})]} \int_{-\infty}^{\infty} \exp\left\{-\frac{n(x - \tilde{x})^2 f''(\tilde{x})}{2}\right\} \mathrm{d}x \end{aligned}$$

Laplace's method: derivation II

- We recognise $-n(x-\tilde{x})^2 f''(\tilde{x})/2$ as the exponential part of the Normal($\tilde{x}$, $1/nf''(\tilde{x})$) distribution, which has pdf

$$\sqrt{\frac{nf''(\tilde{x})}{2\pi}} \exp\left\{-\frac{n(x-\tilde{x})^2 f''(\tilde{x})}{2}\right\}$$

  and must integrate to unity

- So

$$\int_{-\infty}^{\infty} \exp\left\{-\frac{n(x-\tilde{x})^2 f''(\tilde{x})}{2}\right\} \mathrm{d}x = \sqrt{\frac{2\pi}{nf''(\tilde{x})}}$$

- Substituting this into I_n above we get $I_n \simeq \mathrm{e}^{-n[f(\tilde{x})]}\sqrt{(2\pi)/nf''(\tilde{x})}$, as stated in Equation (4.4)
- Knowledge of the derivation of Laplace's method is beyond the scope of MTH3045

Example: Poisson marginal approximation using Laplace's method I

- Recall Example 4.8
- Use Laplace's method to approximate $f(y)$
- We'll calculate

$$f(y) = \int_{-\infty}^{\infty} f(y,\lambda)\mathrm{d}\lambda \simeq \mathrm{e}^{-n[-\log f(y,\tilde{\lambda})]}\sqrt{\frac{2\pi}{ng''(y,\tilde{\lambda})}}$$

where

$$g''(y,\tilde{\lambda}) = -\left.\frac{\partial^2 \log f(y,\lambda)}{\partial\lambda^2}\right|_{\lambda=\tilde{\lambda}}$$

- Given Equation (4.3) we have

$$-\log f(y,\lambda) = \lambda - y\log(\lambda) + \frac{1}{2\sigma^2}(y-\lambda)^2 + \text{constant},$$

with $n = 1$, which is based on swapping x for (y,λ)

- Then, differentiating w.r.t. λ,

$$-\frac{\partial \log f(y,\lambda)}{\partial\lambda} = 1 - \frac{y}{\lambda} - \frac{1}{\sigma^2}(y-\lambda) \quad \text{and} \quad -\frac{\partial^2 \log f(y,\lambda)}{\partial\lambda^2} = \frac{y}{\lambda^2} + \frac{1}{\sigma^2}$$

Example: Poisson marginal approximation using Laplace's method II

- We can't find $\tilde{\lambda}$ analytically, and so we'll find it numerically instead
- The following function, `tilde_lambda(y, mu, sigma)`, gives $\tilde{\lambda}$ given $y =$ `y`, $\mu =$ `mu` and $\sigma =$ `sigma`

```
f <- function(lambda, y, mu, sigma)
  - dpois(y, lambda, log = TRUE) - dnorm(lambda, mu, sigma, log = TRUE)
tilde_lambda <- function(y, mu, sigma) {
  optimize(f, c(.1, 20), y = y, mu = mu, sigma = sigma)$minimum
}
```

- We'll re-use `mu`, `sigma` and `y_vals` from Example 4.8
- We then want to find $\tilde{\lambda}$ for each y in `y_vals`, which we'll store as the vector `tilde_lambdas`

```
tilde_lambdas <- sapply(y_vals, tilde_lambda, mu = mu, sigma = sigma)
tilde_lambdas
```

```
##  [1]  1.000  3.541  4.772  5.720  6.521  7.227  7.865  8.453  9.000
## [10]  9.514 10.000 10.462 10.904 11.328 11.736 12.130 12.510 12.879
## [19] 13.238 13.586 13.926 14.257 14.580 14.896 15.205 15.508 15.805
## [28] 16.096 16.382 16.663 16.939
```

Example: Poisson marginal approximation using Laplace's method III

- We then want to evaluate the approximation to I_n given by Equation (4.4)
- We'll start with a function to evaluate $f''(\tilde{\lambda})$, which we'll use for each $\tilde{\lambda}$ in `tilde_lambdas`

```
f2 <- function(lambda, y, mu, sigma)
  y / lambda^2 + 1 / sigma^2
f2_lambdas <- f2(tilde_lambdas, y_vals, mu, sigma)
f2_lambdas
```

```
##  [1]  1.000  3.541  4.772  5.720  6.521  7.227  7.865  8.453  9.000
## [10]  9.514 10.000 10.462 10.904 11.328 11.736 12.130 12.510 12.879
## [19] 13.238 13.586 13.926 14.257 14.580 14.896 15.205 15.508 15.805
## [28] 16.096 16.382 16.663 16.939
```

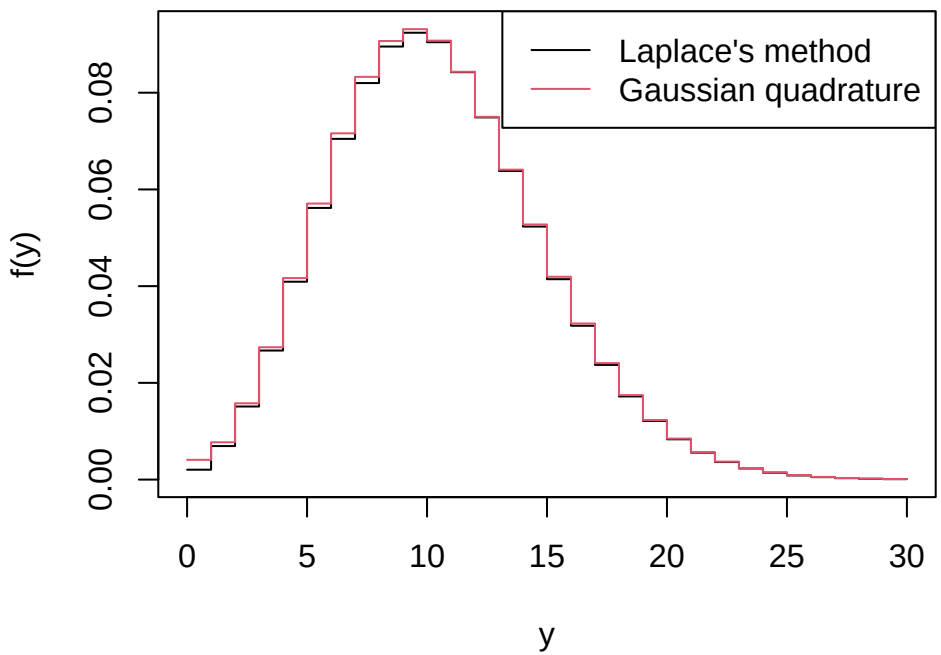
- Then we'll approximate I_n and store this as `fhat2`

```
fhat2 <- sqrt(2 * pi / f2_lambdas) * exp(-f(tilde_lambdas, y_vals, mu, sigma))
```

Example: Poisson marginal approximation using Laplace's method IV

- Finally we'll plot the Laplace approximation to $f(y)$ against y for `y_vals`, i.e. `fhat2`, alongside the estimate obtained by Gaussian quadrature in Example 4.8
  - and see that the two approaches to approximate $f(y)$ give very similar results

```
matplot(y_vals, cbind(fhat, fhat2),
        col = 1:2, lty = 1, type = 'l', xlab = 'y', ylab = 'f(y)')
legend('topright', c("Laplace's method", "Gaussian quadrature"),
       lty = 1, col = 1:2)
```



Laplace's method for multiple dimensions I

- Above we considered Taylor series for a function of one variable
- Laplace's method extends to higher dimensions, although the one-variable case will only be assessed in MTH3045
- Consider the Taylor series for functions of multiple variables, denote $f(\mathbf{x})$, where $f : \mathbb{R}^n \to \mathbb{R}$
- The infinite series notation becomes a bit cumbersome, so we'll skip to the second-order approximation

$$f(\mathbf{x}) \simeq f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^{\mathsf{T}} \nabla f(\mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^{\mathsf{T}} \left[\nabla^2 f(\mathbf{x}_0)\right] (\mathbf{x} - \mathbf{x}_0)$$

Laplace's method for multiple dimensions II

- Then consider the integral

$$I_n = \int \mathrm{e}^{-nf(\mathbf{x})} \mathrm{d}\mathbf{x}.$$

- Laplace's method gives that

$$I_n \simeq \left(\frac{2\pi}{n}\right)^{d/2} \frac{\mathrm{e}^{-nf(\tilde{\mathbf{x}})}}{|\mathbf{H}|^{1/2}},$$

  where $\mathbf{H}$ is the Hessian matrix of $f(\mathbf{x})$ evaluated at $\mathbf{x} = \tilde{\mathbf{x}}$, i.e. $\nabla^2 f(\tilde{\mathbf{x}})$

- The above approximation results from integrating out the $MVN_p(\tilde{\mathbf{x}}, \mathbf{H}^{-1})$ distribution
- This is effectively a multivariate extension to integrating out the $N(\tilde{x}, 1/nf''(\tilde{x}))$, which led to Equation (4.4)

Challenges II

- Go to Challenges II of the week 8 lecture 3 challenges at https://byoungman.github.io/MTH3045/challenges

Challenge II solution I

- We first notice that, given Equation (4.4), we take $f()$ to be $-\log f(y, \lambda)$ and x to be λ, as that's what we're integrating out.
- Then we get $f(y, \lambda) = f(y \mid \lambda) f(\lambda) = \lambda e^{-\lambda y} \times 4\lambda e^{-2\lambda} = 4\lambda^2 e^{-\lambda(2+y)}$
- So $-\log f(y, \lambda) = -2 \log \lambda + \lambda(2 + y) - \log 2$
- Then $-\dfrac{\partial \log f(y, \lambda)}{\partial \lambda} = -\dfrac{2}{\lambda} + (2 + y)$, which is minimised at $\hat{\lambda} = 2/(2 + y)$
- Also $-\dfrac{\partial^2 \log f(y, \lambda)}{\partial \lambda^2} = 2/\lambda^2$ and $2/\hat{\lambda}^2 = 2/(4/(2 + y)^2) = (2 + y)^2/2$
- So, if $g''(\lambda)$ denotes $-\partial^2 \log f(y, \lambda)/\partial \lambda^2$, then

$$I_n = f(y, \tilde{\lambda}) \sqrt{\frac{2\pi}{g''(\tilde{\lambda})}}$$

$$= 4\hat{\lambda}^2 e^{-\hat{\lambda}(2+y)} \sqrt{2\pi \times 2(2 + y)^{-2}} = \frac{32\sqrt{\pi}}{e^2(2 + y)^3}$$

Challenge II solution II

```
par(mar = c(4, 4, 1, 1))
y <- seq(0, 8, by = .1)
plot(y, 32 * sqrt(pi) * exp(-2) / (2 + y)^3,
     ylim = 0:1, xlab = 'y', ylab = 'f(y)')
points(y, 8 / (y + 2)^3, pch = 2, col = 2)
```