

MTH3045: Statistical Computing

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

17/02/2025

Week 6 lecture 1

Challenges I

- Go to Challenges I of the week 6 lecture 1 challenges at <https://byoungman.github.io/MTH3045/challenges>

Sherman-Morrison formula / Woodbury matrix identity

- In general, unless we actually need the inverse of a matrix (such as for standard errors of regression coefficients in a linear model), we should solve systems of linear equations
- Sometimes, though, we do need – or might just have – an inverse, and want to calculate something related to it
- The following are a set of formulae, which go by various names, that can be useful in this situation

Woodbury's formula

- Consider an $m \times m$ matrix \mathbf{A} , with m large, and for which we have the inverse, \mathbf{A}^{-1}
- Suppose \mathbf{A} receives a small update of the form $\mathbf{A} + \mathbf{UV}^T$, for $m \times n$ matrices \mathbf{U} and \mathbf{V} where $n \ll m$
- Then, by **Woodbury's formula**,

$$(\mathbf{A} + \mathbf{UV}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I}_n + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^T\mathbf{A}^{-1}$$

- *Remark:* What's important to note here is that we're looking to calculate an $m \times m$ inverse with m large, and so in general this will be an $O(m^3)$ calculation based on the LHS
- However, in the above, the RHS only requires that we to invert an $n \times n$ matrix, at cost $O(n^3)$, which is much less than that of the LHS, if we already have \mathbf{A}^{-1} .

Sherman-Morrison-Woodbury formula

- Woodbury's formula above generalises to the so-called **Sherman-Morrison-Woodbury formula** by introducing the $n \times n$ matrix **C**, so that

$$(\mathbf{A} + \mathbf{UCV}^T)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}^T\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}^T\mathbf{A}^{-1}$$

Challenges II

- Go to Challenges II of the week 6 lecture 1 challenges at <https://byoungman.github.io/MTH3045/challenges>

Sherman-Morrison formula

- The **Sherman-Morrison formula** is the special case of Woodbury's formula (and hence the Woodbury-Sherman-Morrison formula) in which the update to \mathbf{A} can be considered in terms of m -vectors \mathbf{u} and \mathbf{v} , so that

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}$$

- *Remark:* The Sherman-Morrison formula is particularly useful because it requires no matrix inversion

Example: Bayesian linear regression I

- Recall from MTH2006 the normal linear model where

$$Y_i \sim N(\mathbf{x}_i^T \boldsymbol{\beta}, \sigma^2)$$

with independent errors $\varepsilon_i = Y_i - \mathbf{x}_i^T \boldsymbol{\beta}$, for $i = 1, \dots, n$, where $\mathbf{x}_i^T = (1, x_{i1}, \dots, x_{ip})$ is the i th row of design matrix \mathbf{X} and where $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_p)^T$

- Hence

$$\mathbf{Y} \mid \mathbf{X}\boldsymbol{\beta} \sim MVN_n(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}_n)$$

- In Bayesian linear regression, the elements of $\boldsymbol{\beta}$ and σ^2 are not fixed, unknown parameters: they are random variables, and we must declare *a priori* our beliefs about their distribution
- The conjugate prior is that

$$\boldsymbol{\beta} \sim MVN_{p+1}(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta^{-1})$$

- Integrating out $\boldsymbol{\beta}$ gives

$$\mathbf{Y} \sim MVN_n(\mathbf{X}\boldsymbol{\mu}_\beta, \sigma^2 \mathbf{I}_n + \mathbf{X}\boldsymbol{\Sigma}_\beta^{-1}\mathbf{X}^T)$$

Example: Bayesian linear regression II

- Now suppose that we want to evaluate the marginal likelihood for an observation, \mathbf{y} , say. Recall the MVN pdf
- The Mahalanobis distance then involves the term

$$(\mathbf{y} - \mathbf{X}\boldsymbol{\mu}_\beta)^\top (\sigma^2 \mathbf{I}_n + \mathbf{X}\boldsymbol{\Sigma}_\beta^{-1}\mathbf{X}^\top)^{-1} (\mathbf{y} - \mathbf{X}\boldsymbol{\mu}_\beta)$$

- The covariance of the marginal distribution, $\sigma^2 \mathbf{I}_n + \mathbf{X}\boldsymbol{\Sigma}_\beta^{-1}\mathbf{X}^\top$, is typically dense, expensive to form, and leads to expensive solutions to systems of linear equations
- Its inverse, however, can be computed through the Sherman-Morrison-Woodbury formula with $\mathbf{A}^{-1} = \sigma^{-2} \mathbf{I}_n$, $\mathbf{U} = \mathbf{V} = \mathbf{X}$, and $\mathbf{C} = \boldsymbol{\Sigma}_\beta^{-1}$

Bibliographic notes

- For more details on matrix decompositions, consider Wood (2015, Appendix B) for a concise overview
- For fuller details consider Monahan (2011, chaps. 3, 4 and 6) or Press et al. (2007, chap. 2 and 11)

Chapter 4: Numerical Calculus

Numerical Calculus: Motivation

- In statistics, we often rely on the Normal distribution with pdf

$$\phi(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left[-\frac{(x - \mu)^2}{2\sigma^2} \right]$$

and cdf

$$\Phi(x; \mu, \sigma^2) = \int_{-\infty}^x \phi(u; \mu, \sigma^2) du.$$

- Unfortunately no closed form exists for $\Phi(x; \mu, \sigma^2)$
- However, if x , μ and σ are stored in R as `x`, `mu` and `sigma`, we can still evaluate $\Phi(x; \mu, \sigma^2)$ with `pnorm(x, mu, sigma)`
- This is one example of a frequently-occurring situation in which we somehow want to evaluate an intractable integral
- This raises the question: are there generic methods that let us evaluate intractable integrals?
- The answer is often **numerical integration**

Numerical Calculus: Motivation

- *Remark:* In this chapter, we'll consider generic methods for integration, i.e. that work in many scenarios
- Sometimes, such as evaluating $\Phi(x; \mu, \sigma^2)$, specific algorithms will give more accurate results
- This is what R does for `pnorm()`

Numerical Differentiation

- In Chapter 5 we will cover optimisation of functions, such as numerically finding maximum likelihood estimates when analytical solutions aren't available
- We'll see that supplying derivatives can considerably improve estimation, typically in terms of reducing computation time
- Here we'll cover some useful results in terms of differentiation of matrices, which will later prove useful
- No knowledge of analytical matrix calculus beyond these results will be needed for MTH3045
- The matrix cookbook (Petersen and Pedersen 2012), however, can provide you with a much more thorough set of differentiation rules, should you ever need them

Differentiation definitions I

- **Definition:** Consider $f : \mathbb{R}^n \rightarrow \mathbb{R}$, which we'll consider a function of vector $\mathbf{x} = (x_1, \dots, x_n)^T$
- The **gradient operator**, ∇ , is defined as

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

- Note that in MTH3045 we will have no cause to consider multivariate functions, i.e. to consider $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, for $m > 1$

Differentiation definitions II

- **Definition:** Consider again $f : \mathbb{R}^n \rightarrow \mathbb{R}$
- The **Hessian matrix** is the matrix of second derivatives of f , whereas the gradient operator considered first derivatives, and is given by

$$\nabla^2 f(\mathbf{x}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

- The Hessian matrix plays an important role in statistics, in particular for estimating parameter uncertainty via the Fisher information, which is covered in MTH3028
- In the next chapter, we'll also see that it's important for optimisation

Differentiation rules I

- Now consider $g : \mathbb{R}^n \rightarrow \mathbb{R}$ a function of \mathbf{x} that, for fixed \mathbf{A} , takes the quadratic form $g(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$ for $n \times n$ matrix \mathbf{A} and n -vector \mathbf{x}
- Then

$$\nabla g(\mathbf{x}) = (\mathbf{A} + \mathbf{A}^\top) \mathbf{x} \quad \text{and} \quad \nabla^2 g(\mathbf{x}) = \mathbf{A} + \mathbf{A}^\top$$

- Note that in the case of symmetric \mathbf{A} , $\nabla g(\mathbf{x}) = 2\mathbf{A}\mathbf{x}$ and $\nabla^2 g(\mathbf{x}) = 2\mathbf{A}$

Differentiation rules II

- Next consider $h : \mathbb{R}^n \rightarrow \mathbb{R}$ a function of n -vector \mathbf{x} and p -vector \mathbf{y} that, for fixed $n \times n$ matrix \mathbf{A} and $n \times p$ matrix \mathbf{B} , takes the quadratic form $h(\mathbf{x}, \mathbf{y}) = (\mathbf{x} + \mathbf{B}\mathbf{y})^T \mathbf{A}(\mathbf{x} + \mathbf{B}\mathbf{y})$
- Then

$$\frac{\partial h(\mathbf{x}, \mathbf{y})}{\partial \mathbf{x}} = (\mathbf{A} + \mathbf{A}^T)(\mathbf{x} + \mathbf{B}\mathbf{y}) \quad \text{and} \quad \frac{\partial h(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y}} = \mathbf{B}^T(\mathbf{A} + \mathbf{A}^T)(\mathbf{x} + \mathbf{B}\mathbf{y}),$$

and also

$$\frac{\partial^2 h(\mathbf{x}, \mathbf{y})}{\partial \mathbf{x} \partial \mathbf{x}^T} = \mathbf{A} + \mathbf{A}^T \quad \text{and} \quad \frac{\partial^2 h(\mathbf{x}, \mathbf{y})}{\partial \mathbf{y} \partial \mathbf{y}^T} = \mathbf{B}^T(\mathbf{A} + \mathbf{A}^T)\mathbf{B}$$

- Note that above we use partial derivative notation, i.e. ∂ , as opposed to gradient operator notation, i.e. ∇ , as the derivatives are not w.r.t. all variables.

Challenges III

- Go to Challenges III of the week 6 lecture 1 challenges at <https://byoungman.github.io/MTH3045/challenges>

Week 6 lecture 2

Example: Maximum likelihood estimates of regression coefficients in the normal linear model via matrix calculus I

- Consider the normal linear model $\mathbf{Y} = \mathbf{X}\beta + \varepsilon$
 - $\mathbf{Y} = (Y_1, \dots, Y_n)^\top$
 - \mathbf{X} is an $n \times (p + 1)$ design matrix
 - β is a $(p + 1)$ -vector of regression coefficients
 - $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n)^\top$ with independent $\varepsilon_i \sim N(0, \sigma^2)$
- The maximum likelihood estimate of β , denoted $\hat{\beta}$, minimises the RSS, i.e. minimises

$$(\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta)$$

if we observe $\mathbf{y} = (y_1, \dots, y_n)^\top$

- Differentiating w.r.t. β gives

$$-\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\beta) - (\mathbf{y} - \mathbf{X}\beta)^\top \mathbf{X}$$

Example: Maximum likelihood estimates of regression coefficients in the normal linear model via matrix calculus II

- Then

$$-\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) - (\mathbf{y} - \mathbf{X}\beta)^T\mathbf{X}$$

simplifies to

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)$$

as $\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)$ and $(\mathbf{y} - \mathbf{X}\beta)^T\mathbf{X}$ are both n -vectors

- The derivative of the RSS is zero at $\hat{\beta}$ and so

$$-2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\beta}) = 0$$

- Therefore $\hat{\beta}$ is the solution of

$$\mathbf{X}^T\mathbf{X}\hat{\beta} = \mathbf{X}^T\mathbf{y}$$

- Alternatively

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

as we were given in MTH2006

Finite-differencing

- Consider again $f(\mathbf{x})$, a function of vector \mathbf{x}
- **Definition:** Consider $f : \mathbb{R}^n \rightarrow \mathbb{R}$ for n -vector \mathbf{x}
- Let, \mathbf{e}_i be the n -vector comprising entirely zeros, except for its i th element, which is one
- Then the **partial derivative** of $f(\mathbf{x})$ w.r.t. x_i , the i th element of \mathbf{x} , is

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}$$

- The preceding definition leads us to the **finite-difference** partial derivative approximation

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \simeq \frac{f(\mathbf{x} + \delta\mathbf{e}_i) - f(\mathbf{x})}{\delta},$$

where δ is small

Example: Finite-differencing of $\sin(x)$ I

- Use finite-differencing to approximate the derivative of $f(x) = \sin(x)$ for $x \in [0, 2\pi]$, and compare its accuracy to the true derivative
- First, let's calculate f and its *true* derivative, i.e. $f'(x) = \cos(x)$, and store this as `partial0` for $\{x_i\}$, $i = 1, \dots, 100$, a set of equally-spaced points on $[0, 2\pi]$

```
n <- 100
x <- seq(0, 2 * pi, l = 100)
f <- sin(x)
partial0 <- cos(x)
```

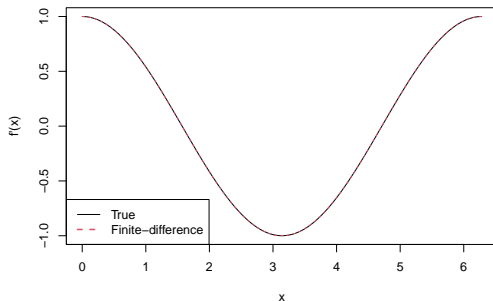
- Now we'll set $\delta = 10^{-6}$ and calculate $x_i + \delta$, for each x_i , i.e. each element of `x`, so that the finite-difference approximation to the derivative is given by $[\sin(x_i + 10^{-6}) - \sin(x_i)]/10^{-6}$, which is calculated below and stored as `partial1`

```
delta1 <- 1e-6
x1 <- x + delta1
f1 <- sin(x1)
partial1 <- (f1 - f) / delta1
```

Example: Finite-differencing of $\sin(x)$ II

- We'll then plot the $f'(x)$ against its finite-difference approximation

```
matplot(x, cbind(partial0, partial1), type = 'l', xlab = 'x', ylab = "f'(x)")  
legend('bottomleft', lty = 1:2, col = 1:2, lwd = 1:2,  
       legend = c('True', 'Finite-difference'))
```



- In fact, the true derivative and its finite-difference approximation are so similar that it's difficult to distinguish the two, but they're both there in the plot!

Example: Finite-differencing of $\sin(x)$ III

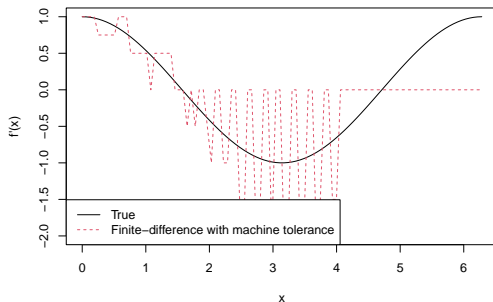
- We might be tempted to choose δ as small as possible
- Suppose we were to repeat Example 4.3 with $\delta = \epsilon_m$, i.e. R's machine tolerance
- The following calculates the finite-difference approximation as `partial2`

```
delta2 <- .Machine$double.eps  
x2 <- x + delta2  
f2 <- sin(x2)  
partial2 <- (f2 - f) / delta2
```

Example: Finite-differencing of $\sin(x)$ III

- The the following plots this against the true value of $f'(x)$

```
matplot(x, cbind(partial0, partial2), type = 'l', xlab = 'x', ylab = "f'(x)")  
legend('bottomleft', lty = 1:2, col = 1:2, bg = 'white',  
       legend = c('True', 'Finite-difference with machine tolerance'))
```



- Using $\delta = \epsilon_m$ gives a terrible approximation to $f'(x)$, which gets worse as x increases
- We've actually encountered an example *calculation error*, which was introduced in Chapter 2

Challenges I

- Go to Challenges I of the week 6 lecture 2 challenges at <https://byoungman.github.io/MTH3045/challenges>

Example: Finite differencing of the multivariate Normal log-likelihood I

- Find $\partial \log f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) / \partial y_i$ analytically, for $i = 1, \dots, p$, where $f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the $MVN_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ pdf, for arbitrary \mathbf{y} , $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$
- Evaluate this for \mathbf{y} , $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as in Example 3.2
- Then approximate the same derivative using finite-differencing
- The logarithm of the $MVN_p(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ pdf is given by

$$\log f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{1}{2} [p \log(2\pi) + \log(|\boldsymbol{\Sigma}|) + (\mathbf{y} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{y} - \boldsymbol{\mu})]$$

which we know, from Example 3.13, we can evaluate with `dmvn3()`

- Using the above properties

$$\frac{\partial \log f(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \mathbf{y}} = -\boldsymbol{\Sigma}^{-1}(\mathbf{y} - \boldsymbol{\mu})$$

since $\boldsymbol{\Sigma}$ is symmetric

Example: Finite differencing of the multivariate Normal log-likelihood II

- We can evaluate this for \mathbf{y} , $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as in Example 3.2 with the following

```
y <- c(.7, 1.3, 2.6)
mu <- 1:3
Sigma <- matrix(c(4, 2, 1, 2, 3, 2, 1, 2, 2), 3, 3)
deriv1 <- -solve(Sigma, y - mu)
```

which gives

```
as.vector(deriv1)
```

```
## [1] -0.08  0.38 -0.14
```


Example: Finite differencing of the multivariate Normal log-likelihood III

- To approximate the derivative by finite-differencing, it makes sense to write a multi-purpose function for finite differencing, which we'll call `fd()`

```
fd <- function(x, f, delta = 1e-6, ...) {  
  # Function to evaluate derivative w.r.t. vector by finite-differencing  
  # x is a p-vector  
  # fn is the function for which the derivative is being calculated  
  # delta is the finite-differencing step, which defaults to 10^{-6}  
  # returns a vector of length x  
  f0 <- f(x, ...)  
  p <- length(x)  
  f1 <- numeric(p)  
  for (i in 1:p) {  
    ei <- replace(numeric(p), i, 1)  
    f1[i] <- f(x + delta * ei, ...)  
  }  
  (f1 - f0) / delta  
}
```

Example: Finite differencing of the multivariate Normal log-likelihood IV

- Then we can use this with `dmvn3()` with the following

```
deriv2 <- fd(y, dmvn3, mu = mu, Sigma = Sigma)
```

which gives

```
deriv2
```

```
## [1] -0.0800002  0.3799993 -0.1400008
```

and is the same as the analytical result

```
all.equal(deriv1, deriv2)
```

```
## [1] "Mean relative difference: 2.832689e-06"
```

once we allow for error in the finite-difference approximation

Challenges II

- Go to Challenges II of the week 6 lecture 2 challenges at <https://byoungman.github.io/MTH3045/challenges>

Week 6 lecture 3

Quadrature

- Another common requirement in statistics is that some integral needs to be evaluated
- To start, let's consider a simple integral of the form

$$I = \int_a^b f(x)dx$$

- We'll first take a look at some *deterministic* approaches to numerically evaluating integrals
- In fact, these all boil down to assuming that

$$I \simeq \sum_{i=1}^N w_i f(x_i^*)$$

for some weights w_i and nodes x_i^* , $i = 1, \dots, N$

- Note that here we're considering the so-called *composite* approach to approximating an interval, i.e. in which a rule is applied over a collection of sub-intervals

Midpoint rule

- Perhaps the first numerical integration scheme we come across is the midpoint rule
- Put simply, we divide $[a, b]$ into N equally-sized intervals, and use the midpoints of these as the nodes, x_i^*
- This gives

$$\int_a^b f(x)dx \simeq h \sum_{i=1}^N f(x_i^*),$$

where $x_i^* = a + (i - 0.5)(b - a)/N$ and $h = (b - a)/N$

- The error in the approximation is $O(h^2)$
- Thus more intervals reduces h and gives a more accurate approximation
- We can measure accuracy through **relative absolute error**
- **Definition:** The **relative absolute error**, or sometimes just *relative error*, of an estimate of some true value is given by

$$\left| \frac{\text{true value} - \text{estimate}}{\text{true value}} \right|$$

Example: Midpoint rule I

- Consider the integral $\int_0^1 \exp(x) dx = \exp(1) - 1 \simeq 1.7182818$
- Use R and the midpoint rule to estimate the integral with $N = 10, 100$ and 1000
- Then compare the relative absolute error of each
- We'll start by calculating the true value of the integral, which we'll store as `true`

```
true <- exp(1) - 1
```

- Then we'll store the values of N that we're testing as `N_vals`

```
N_vals <- 10^c(1:3)
```

Example: Midpoint rule II

- The following then creates a vector, `midpoint`, in which to store the integral approximations, and calculates the approximations with a `for` loop
- Inside the loop the integration nodes (i.e. the midpoints) and h are calculated

```
midpoint <- numeric(length(N_vals))
for (i in 1:length(N_vals)) {
  N <- N_vals[i]
  nodes <- (1:N - .5) / N
  h <- 1 / N
  midpoint[i] <- h * sum(exp(nodes))
}
midpoint
```

```
## [1] 1.717566 1.718275 1.718282
```


Example: Midpoint rule III

- The relative absolute error for each is then given in the vector `rel_err_mp` below

```
rel_err_mp <- abs((true - midpoint) / true)
rel_err_mp
```

```
## [1] 4.165452e-04 4.166655e-06 4.166667e-08
```

- We clearly see that the absolute error reduces by two factors of ten for each factor of ten increase in N , which is consistent with the above comment of $O(h^2)$ error, where here $h = 1/N$

Challenges I

- Go to Challenges I of the week 6 lecture 3 challenges at <https://byoungman.github.io/MTH3045/challenges>

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

$$\int_a^b f(x)dx \simeq \frac{h}{6} \left(f(a) + 4 \sum_{i=1}^N f(x_{1i}^*) + 2 \sum_{i=1}^{N-1} f(x_{2i}^*) + f(b) \right),$$

where $x_{1i}^* = a + h(2i - 1)/2$, $x_{2i}^* = a + ih$ and $h = (b - a)/N$

- Note that Simpson's rule requires $N + 1$ more evaluations of f than the midpoint rule
 - however, a benefit of those extra evaluations 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 $\int_0^1 \exp(x) dx = \exp(1) - 1$ 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 create two sets of nodes, `nodes1` and `nodes2`, which correspond to the x_{1i} s and x_{2i} s in Equation (4.1), respectively

Example: Simpson's rule II

- The integral approximations are stored as `simpson`

```
simpson <- numeric(length(N_vals))
N_vals <- 10^c(1:3)
for (i in 1:length(N_vals)) {
  N <- N_vals[i]
  h <- 1 / N
  simpson[i] <- 1 + exp(1)
  nodes1 <- h * (2*c(1:N) - 1) / 2
  simpson[i] <- simpson[i] + 4 * sum(exp(nodes1))
  nodes2 <- h * c(1:(N - 1))
  simpson[i] <- simpson[i] + 2 * sum(exp(nodes2))
  simpson[i] <- h * simpson[i] / 6
}
print(simpson, digits = 12)
```

```
## [1] 1.71828188810 1.71828182847 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] 3.471189e-08 3.472270e-12 6.461239e-16
```

- 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 from 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 II

- Go to Challenges II of the week 6 lecture 3 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

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

where, for $i = 1, \dots, N$, 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

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

for a fixed set of x values, x_i with corresponding weights w_i , for $i = 1, \dots, N$

- 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
```

```
##
```

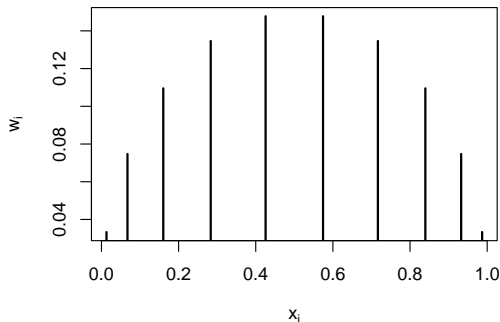
```
## $w
```

```
## [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 polynomials¹

¹Wikipedia 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 $\int_0^1 \exp(x) dx$ 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`,

```
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 III

- Go to Challenges III of the week 6 lecture 3 challenges at <https://byoungman.github.io/MTH3045/challenges>

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 $\lambda \sim N(\mu, \sigma^2)$
- 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

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

- *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 $N(10, 3^2)$ 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, \dots, 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 $0, 1, \dots, 30$, which we can create in R with

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

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

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

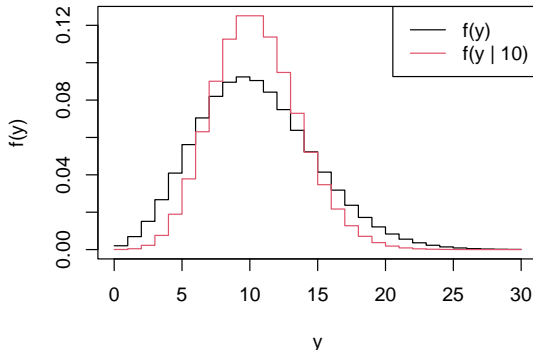
- The following code gives $\hat{f}(y)$ as `fhat` for y in the set of values `y_vals`

```
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 $f(y | 10)$, which is to be expected given that $f(y)$ integrates out the variability in λ given by the $N(10, 3^2)$ 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

References

- Monahan, John F. 2011. *Numerical Methods of Statistics*. 2nd ed. Cambridge University Press.
<https://doi.org/10.1017/CBO9780511977176>.
- Petersen, K. B., and M. S. Pedersen. 2012. *The Matrix Cookbook*.
<https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf>.
- Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. 2007. *Numerical Recipes: The Art of Scientific Computing*. 3rd ed. Cambridge University Press.
<https://books.google.co.uk/books?id=1aAOdzK3FegC>.
- Wood, Simon N. 2015. *Core Statistics*. Institute of Mathematical Statistics Textbooks. Cambridge University Press.
<https://doi.org/10.1017/CBO9781107741973>.