Final Exam

Michael Nelson

Problem 1

Exercise 1. You are given the quadrature formula

$$Q_*(f,a,b) = \left(\frac{b-a}{2}\right) f\left(\frac{3}{4}a + \frac{1}{4}b\right) + \left(\frac{b-a}{2}\right) f\left(\frac{1}{4}a + \frac{3}{4}b\right)$$

where $a \neq b$.

1. Determine to what degree m this formula integrates polynomials exactly.
2. Compute the approximation of
$$\int_0^1 \frac{1}{\sqrt{x}} \mathrm{d}x$$
using Q_* as given above (simplify your result, it may contain expressions like $\sqrt{2}$ and $\sqrt{3}$).
3. How does the quadrature Q_* compare to Newton-Cotes and Gauss rules? Which one of the three would you use in practice?

Solution 1. 1. We claim that Q_* is of degree 2, meaning it integrates polynomials of degree ≤ 1 exactly but that there exists a polynomial of degree 2 for which it does not integrate exactly. Indeed, first note that Q_* integrates the constant function 1 exactly:

$$\begin{aligned} Q_*(1,a,b) &= \left(\frac{b-a}{2}\right) \cdot 1 + \left(\frac{b-a}{2}\right) \cdot 1 \\ &= b-a \\ &= \int_a^b \mathrm{d}x. \end{aligned}$$

Next note that Q_* integrates the function x exactly:

$$\begin{aligned} Q_*(x,a,b) &= \left(\frac{b-a}{2}\right)\left(\frac{3}{4}a + \frac{1}{4}b\right) + \left(\frac{b-a}{2}\right)\left(\frac{1}{4}a + \frac{3}{4}b\right) \\ &= \left(\frac{b-a}{2}\right)\left(\frac{3}{4}a + \frac{1}{4}b + \frac{1}{4}a + \frac{3}{4}b\right) \\ &= \left(\frac{b-a}{2}\right)(b+a) \\ &= \frac{b^2-a^2}{2} \\ &= \int_a^b x\mathrm{d}x. \end{aligned}$$

It follows that Q_* integrates all polynomials of degree ≤ 1 exactly since Q_* is linear in the first argument:

$$\begin{aligned} Q_*(c_0 + c_1 x, a, b) &= c_0 Q_*(1,a,b) + c_1 Q_*(x,a,b) \\ &= c_0 \int_a^b \mathrm{d}x + \int_a^b x\mathrm{d}x \\ &= \int_a^b (c_0 + c_1 x)\mathrm{d}x. \end{aligned}$$

Now let us see that there exists a polynomial of degree 2 for which it does not integrate exactly. Set $x_1 = (3/4)a + (1/4)b$, set $x_2 = (1/4)a + (3/4)b$, set $w = (b-a)/2$, and consider the polynomial $p(x) = 3(x-x_1)^2$. Then observe that on the one hand, we have

$$\begin{aligned} Q_*(p,a,b) &= wp(x_1) + wp(x_2) \\ &= wp(x_2) \\ &= 3w(x_2 - x_1)^2 \\ &= 3w\left(\frac{b-a}{2}\right)^2 \\ &= 3\left(\frac{b-a}{2}\right)^3 \\ &= \frac{3}{8}(b-a)^3. \end{aligned}$$

On the other hand, observe that

$$\begin{aligned} \int_a^b p(x)\mathrm{d}x &= \int_a^b 3(x-x_1)^2 \mathrm{d}x \\ &= \int_{a-x_1}^{b-x_1} 3u^2 \mathrm{d}u \\ &= u^3 \Big|_{a-x_1}^{b-x_1} \\ &= (b-x_1)^3 - (a-x_1)^3 \\ &= \left(\frac{3(b-a)}{4}\right)^3 - \left(\frac{a-b}{4}\right)^3 \\ &= \left(\frac{3^3+1}{4^3}\right)(b-a)^3 \\ &= \frac{7}{16}(b-a)^3. \end{aligned}$$

Since $3/8 \neq 7/16$, we see that $Q_*(p,a,b) \neq \int_a^b p(x)\mathrm{d}x$. Thus Q_* does not integrate the degree 2 polynomial p exactly; hence Q_* has degree 1.

2. We calculate

$$\begin{aligned} Q_*(x^{-1/2}, 0, 1) &= \frac{1}{2}\left(\frac{1}{4}\right)^{-1/2} + \frac{1}{2}\left(\frac{3}{4}\right)^{-1/2} \\ &= \frac{1}{2} \cdot 2 + \frac{1}{2} \cdot \frac{2}{\sqrt{3}} \\ &= 1 + \frac{1}{\sqrt{3}}. \end{aligned}$$

3. The 2-point Newton Cotes quadrature rule is also known at the **trapezoid rule**. It is given by

$$T(f,a,b) = \frac{b-a}{2}(f(a) + f(b))$$

and it is of degree 1. This rule cannot be used to approximate the integral of $x^{-1/2}$ from 0 to 1 however. The 1-point Newton Cotes quadrature rule is also known as the **midpoint rule**. It is given by

$$M(f,a,b) = (b-a)f\left(\frac{a+b}{2}\right)$$

and it is also of degree 1. We can use this rule to approximate the integral of $x^{-1/2}$ from 0 to 1:

$$\begin{aligned} M(x^{-1/2}, 0, 1) &= (1-0)\left(\frac{0+1}{2}\right)^{-1/2} \\ &= 1 \cdot \left(\frac{1}{2}\right)^{-1/2} \\ &= \sqrt{2}. \end{aligned}$$

Since $\int_0^1 x^{-1/2} \mathrm{d}x = 2$, we see that Q_* gives a better approximation of this integral than M. Finally, in Gaussian quadrature, both nodes and weights are optimally chosen to maximize the degree of the quadrature rule. In particular, there is a unique 2-point Gaussian rule of degree 3. In practice, we would want to use this rule since it gives the best approximation.

Problem 2

Exercise 2. We consider the following time stepping methods for an initial value problem $y' = f(t, y)$ with time step size $h = t_{k+1} - t_k$:

- Method A:
$$y_{k+1} = y_k + hf\left(t_k + \frac{h}{2}, \frac{y_k + y_{k+1}}{2}\right).$$
- Method B:
$$y_{k+1} = y_k + hf\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} f(t_k, y_k)\right)$$

1. Determine the stability regions for the methods above using the test equation $y' = \lambda y$.
2. Implement both methods with the signature function y=methodA(func,t,y1) (like forward/backward Euler in class).
3. Using the problem
$$y'(t) = 10\cos t - 2y(t)$$
with $y(0) = 1$ and $0 \leq t \leq 1$ solve the ODE using forward Euler, method A, and method B for different number of time steps $n = 2^4, 2^5, \ldots, 2^9$ and compute the error to the reference solution. The error is defined as
$$\max_{1 \leq k \leq n} |y_{\text{ref}}(t_k) - y_k|$$
if $y_{\text{ref}}(t_k)$ is the exact solution at time t_k. Output a table for each of the three methods with three columns each: time step size, corresponding error, and error rate. Note that the exact solution to this ODE is given by
$$y(t) = -3e^{-2t} + 2\sin t + 4\cos t.$$

Solution 2. 1. First we apply method A to the test ODE $y' = \lambda y$. We obtain

$$\begin{aligned} y_{k+1} &= y_k + h\lambda\left(\frac{y_k + y_{k+1}}{2}\right) \\ &= \left(1 + \frac{h\lambda}{2}\right) y_k + \frac{h\lambda}{2} y_{k+1} \\ &= \left(\frac{2 + h\lambda}{2}\right) y_k + \frac{h\lambda}{2} y_{k+1}. \end{aligned}$$

We can re-express this as

$$\begin{aligned} y_{k+1} &= \left(\frac{2 + h\lambda}{2 - h\lambda}\right) y_k \\ &= \left(\frac{2 + h\lambda}{2 - h\lambda}\right)^k y_0. \end{aligned}$$

Thus, writing λ in terms of its real an imaginary parts as $\lambda = \lambda_1 + i\lambda_2$, we see that

$$\begin{aligned}
\text{method A is stable} &\iff \left|\frac{2+h\lambda}{2-h\lambda}\right| \leq 1 \\
&\iff |2+h\lambda| \leq |2-h\lambda| \\
&\iff |2+h\lambda|^2 \leq |2-h\lambda|^2 \\
&\iff (2+h\lambda_1)^2 + (h\lambda_2)^2 \leq (2-h\lambda_1)^2 + (-h\lambda_2)^2 \\
&\iff (2+h\lambda_1)^2 \leq (2-h\lambda_1)^2 \\
&\iff |2+h\lambda_1| \leq |2-h\lambda_1|.
\end{aligned}$$

In particular, methodA is stable for all $h > 0$ whenever $\lambda_1 < 0$.

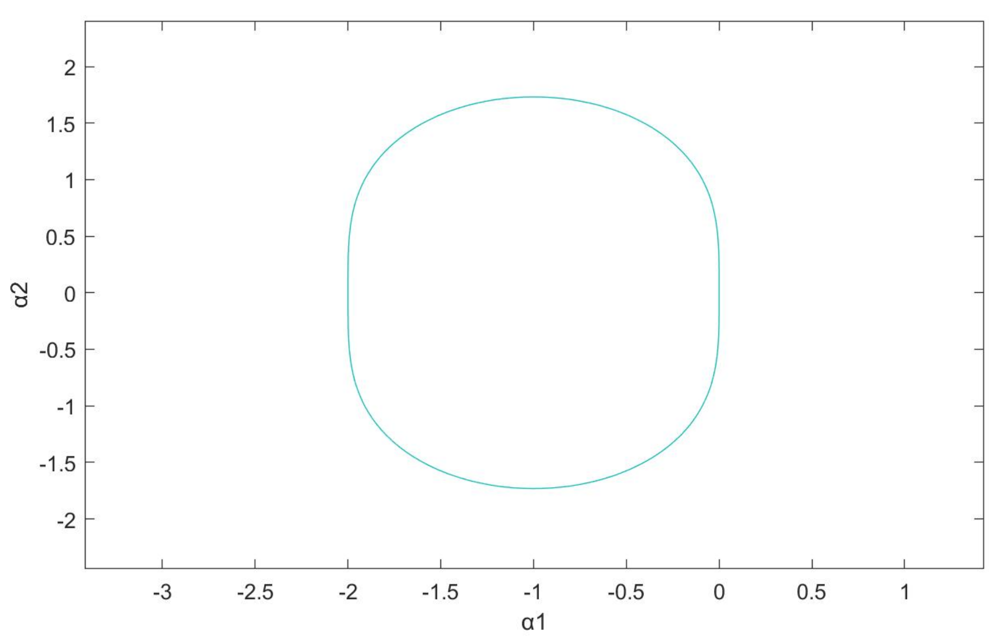
Next we apply method B to the test function $y' = \lambda y$. We obtain

$$\begin{aligned}
y_{k+1} &= y_k + h\lambda\left(y_k + \frac{h}{2}\lambda y_k\right) \\
&= \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2\right) y_k \\
&= \left(1 + h\lambda + \frac{1}{2}(h\lambda)^2\right)^k y_0
\end{aligned}$$

Thus, writing $\alpha = h\lambda$ and $\alpha = \alpha_1 + i\alpha_2$ (so $\alpha_1 = h\lambda_1$ and $\alpha_2 = h\lambda_2$) we see that

$$\begin{aligned}
\text{method B is stable} &\iff \left|1 + \alpha + \frac{1}{2}\alpha^2\right| \leq 1 \\
&\iff \left|1 + \alpha + \frac{1}{2}\alpha^2\right|^2 \leq 1 \\
&\iff \left|(1 + \alpha_1 + \frac{1}{2}(\alpha_1 - \alpha_2)(\alpha_1 + \alpha_2) + i(\alpha_1 + 1)\alpha_2\right|^2 \leq 1 \\
&\iff ((1 + \alpha_1 + \frac{1}{2}(\alpha_1 - \alpha_2)(\alpha_1 + \alpha_2))^2 + ((\alpha_1 + 1)\alpha_2)^2 \leq 1.
\end{aligned}$$

In particular, method B is stable if and only if the point $(\alpha_1, \alpha_2) = h(\lambda_1, \lambda_2)$ lands inside the region bounded by the curve below:



2. First we give the code for method A:

```
function y=methodA(func,t,y1)
% t = [t1,t2,...,tn] has n points and n-1 steps

n = length(t);
y = 0 * t;
y(1)=y1;

for k=1:n-1
   h = t(k+1) - t(k);
   ode_eqn = @(ynext) ynext - y(k) - h * func(t(k)+h/2,(ynext + y(k))/2);
   y(k+1) = fzero(ode_eqn,y(k));
end;
```

Next we give the code for method B:

```
function y=methodB(func,t,y1)
% t = [t1,t2,...,tn] has n points and n-1 steps

n = length(t);
y = 0 * t;
y(1)=y1;

for k=1:length(y)-1
   h = t(k+1) - t(k);
   fk = func(t(k),y(k));
   y(k+1) = y(k) + h * func(t(k)+ (h/2),y(k)+(h/2)*fk);
end;
```

3. Working in matlab, we write:

```
% define functions for this problem

func = @(t,y) 10*cos(t)-2*y;
funcref = @(t) -3*exp(-2*t)+2*sin(t)+4*cos(t);


% define n time steps as vector t=t(n) where t=[t1,t2,...,tn,tn+1] where t1=0 and tn+1=1

t = @(n) 0:2^(-n):1;


% define initial value y1=y(0)=1

y1 = 1;


% create vectors of length n+1 for each method containing approximate solutions

yA = @(n) methodA(func,t(n),y1);
yB = @(n) methodB(func,t(n),y1);
yFE = @(n) forwardEuler(func,t(n),y1);


% create vector of length n+1 containing exact solutions

yref = @(n) funcref(t(n));


% calculate errors for each method using norm(-,Inf)

errorA = @(n) norm(yA(n)-yref(n),Inf);
errorB = @(n) norm(yB(n)-yref(n),Inf);
errorFE = @(n) norm(yFE(n)-yref(n),Inf);


% calculate errorrates for each method from n=4 to n=12

errorratesA=[];
errorratesB=[];
errorratesFE=[];
for n=4:12
        errorrateA = errorA(n+1)/errorA(n);
        errorrateB = errorB(n+1)/errorB(n);
        errorrateFE = errorFE(n+1)/errorFE(n);
        errorratesA = [errorratesA errorrateA];
        errorratesB = [errorratesB errorrateB];
        errorratesFE = [errorratesFE errorrateFE];
end;
```

Now we plot the table for method A:

```
format longg
for n=4:9
        disp([2^(-n) errorA(n) errorratesA(n)]);
end

0.0625          0.00297363505250337           0.249996916651961
0.03125         0.000743128606277121            0.24999922925091
0.015625         0.000185748409893716           0.249999806743112
0.0078125         4.64371189869972e-05           0.250000008262771
0.00390625         1.16091365649496e-05             0.2499999957156
0.001953125         2.90227519350594e-06             0.249999958380116
```

Next we plot the table for method B:

```
for n=4:9
   disp([2^(-n) errorB(n) errorratesB(n)]);
end

0.0625           0.0046803647542335           0.248717212806086
0.03125          0.00112281854007401           0.249359372054863
0.015625          0.00027492791863537             0.24967956766904
0.0078125         6.80259768404134e-05             0.249839852495094
0.00390625           1.6919231358159e-05             0.249919911990386
0.001953125          4.21896890712148e-06                 0.2499599771328
```

Next we plot the table for the forward Euler method:

```
for n=4:9
   disp([2^(-n) errorFE(n) errorratesFE(n)]);
end

0.0625            0.120794793141668           0.498877300734096
0.03125            0.059282752942563           0.499439812045252
0.015625               0.0293702988322           0.499720236721317
0.0078125           0.0146182565026653            0.499860143574655
0.00390625          0.00729271634548834             0.49993007806035
0.001953125           0.00364227288089003             0.499965041753024
```

Problem 3

Exercise 3. Consider the boundary value problem

$$u''(x) + u'(x) = f(x) \tag{1}$$

where $0 \leq x \leq 1$ with $u(0) = u_a$ and $u(1) = u_b$.

1. Derive the linear system for a finite difference approximation of $u(x)$ at points $x_0 = 0, \ldots, x_{n+1} = 1$. Hint: be very careful when deriving the correct terms for u_a and u_b in the right-hand side.

2. Write a function to compute the finite difference solution $y_0, \ldots, y_{n+1}$ at points $x_0, \ldots, x_{n+1}$ to the problem above. The function should be defined as function [X,Y] = finite_difference_solution (n,func,ua,ub) where func is a function handle representing $f(x)$.

3. Compute the finite difference solution for $n = 10$, $u_a = 1$, $u_b = -1$, and

   $$f(x) = -2 - 4\pi(x+1)\sin(2\pi x^2) - 16\pi^2 x^2 \cos(2\pi x^2)$$

   and plot the result. Also include the exact solution $u(x) = \cos(2\pi x^2) - 2x$.

Solution 3. 1. For each $0 \leq k \leq n+1$, we set $y_k = u(x_k)$ and we set $f_k = f(x_k)$. We assume the points $x_0, x_1, \ldots, x_n, x_{n+1}$ For each $1 \leq k \leq n$ we obtain from Taylor's theorem a real-valued function ψ_k defined on a neighborhood of 0 such that $\lim_{h \to 0} \psi_k(h) = 0$ and such that

$$y_{k+1} = y_k + u'(x_k)h + \frac{u''(x_k)}{2}h^2 + \frac{u'''(x_k)}{6}h^3 + \psi_k(h)h^3 \tag{2}$$

and

$$y_{k-1} = y_k - u'(x_k)h + \frac{u''(x_k)}{2}h^2 - \frac{u'''(x_k)}{6}h^3 - \psi_k(-h)h^3 \tag{3}$$

Adding (3) and (2) together and rearranging terms gives us

$$u''(x_k) = \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} + R_k(h) \tag{4}$$

where $R_k(h) = (\psi_k(h) - \psi_k(-h))h$. In particular we have $R_k(h) \in O(h^2)$, thus we may rewrite (4) as

$$u''(x_k) = \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} + O(h^2)$$

In homework 1, it was shown that

$$u'(x_k) = \frac{y_{k+1} - y_{k-1}}{2h} + O(h^2).$$

Thus the discrete version of (1) is given by the equations

$$\frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} + \frac{y_{k+1} - y_{k-1}}{2h} = f_k. \tag{5}$$

or each $1 \leq k \leq n$ with $y_0 = u_a$ and $y_{n+1} = u_b$. We wish to view (5) as an $n \times n$-matrix equation. To do this, first we set $\alpha = \frac{1}{h^2} - \frac{1}{2h}$, $\beta = -\frac{2}{h^2}$, and $\gamma = \frac{1}{h^2} + \frac{1}{2h}$. Then note that we can list the equations (5) starting from $k = 1$ up to $k = n$ as

$$\begin{aligned}
\beta y_1 + \gamma y_2 &= f_1 - \alpha u_a \\
\alpha y_1 + \beta y_2 + \gamma y_3 &= f_2 \\
&\vdots \\
\alpha y_{k-1} + \beta y_k + \gamma y_{k+1} &= f_k \\
&\vdots \\
\alpha y_{n-2} + \beta y_{n-1} + \gamma y_n &= f_{n-1} \\
\alpha y_{n-1} + \beta y_n &= f_n - \gamma u_b
\end{aligned}$$

In matrix form, this looks like

$$\begin{pmatrix}
\beta & \gamma & 0 & 0 & 0 & \cdots & \cdots & 0 & 0 \\
\alpha & \beta & \gamma & 0 & 0 & \cdots & \cdots & 0 & 0 \\
0 & \alpha & \beta & \gamma & 0 & \cdots & \cdots & 0 & 0 \\
0 & 0 & \ddots & \ddots & \ddots & \ddots & & \vdots & \vdots \\
\vdots & \vdots & \ddots & \alpha & \beta & \gamma & \ddots & \vdots & \vdots \\
\vdots & \vdots & & \ddots & \ddots & \ddots & \ddots & 0 & 0 \\
0 & 0 & \cdots & \cdots & 0 & \alpha & \beta & \gamma & 0 \\
0 & 0 & \cdots & \cdots & 0 & 0 & \alpha & \beta & \gamma \\
0 & 0 & \cdots & \cdots & 0 & 0 & 0 & \alpha & \beta
\end{pmatrix}
\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_k \\ \vdots \\ y_{n-2} \\ y_{n-1} \\ y_n \end{pmatrix}
=
\begin{pmatrix} f_1 - \alpha u_a \\ f_2 \\ f_3 \\ \vdots \\ f_k \\ \vdots \\ f_{n-2} \\ f_{n-1} \\ f_n - \gamma u_b \end{pmatrix}.$$

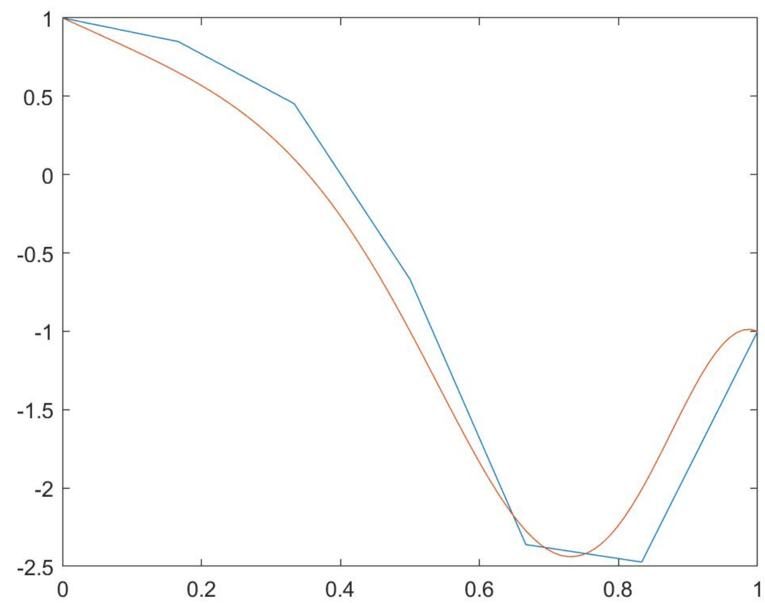
2. We give the code below:

```
function [Y,X] = finite_difference_solution(n,func,ua,ub)

% set step size and create vector X = [x0,x1,...,x{n+1}] where x0=0 and x{n+1}=1

h = 1/(n+1);
X = 0:h:1;


% define alpha, beta, and gamma

alpha = (1/h^2) - (1/2*h);
beta = -2/(h^2);
gamma = (1/h^2) + (1/2*h);


% define vector f = [f1,f2,...,fn]

f = zeros(n,1);
f(1) = func(1/(n+1))-alpha*ua;
f(n) = func(n/(n+1))-gamma*ub;
for k=2:n-1
   f(k)=func(k/(n+1));
end;


% define vector Y = [y1,y2,...,yn]

A = diag(alpha*ones(1,n-1),-1) + diag(beta*ones(1,n)) + diag(gamma*ones(1,n-1),1);
Y = A\f;


% adjoin ua and ub to Y so that Y = [ua,y1,y2,...,yn,ub]

Y = [ua; Y; ub];
```

3. We do this in the code below:

```
func = @(x) -2 - 4*pi*(x+1)*sin(2*pi*x^2) - 16*pi^2*x^2*cos(2*pi*x^2);
funcref = @(x) cos(2*pi.*x.*x)-2.*x;
n=10;
ua=1;
ub=-1;
[Y,X] = finite_difference_solution(n,func,ua,ub);
x = linspace(0,1);
plot(X,Y,x,funcref(x));


x=linspace(0,1);
t=(1/11):(1/11):(10/11);
plot(x,funcref(x),t,u);
```

The plot matlab generates is given below:


1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
0.2
0.4
0.6
0.8
1