University of Nottingham
School of Mathematical Sciences

MATH4063/G14SCC | Scientific Computing and C++

Deadline: 8th January 2021, 3:00pm (GMT) | **Coursework 2 – Solution Template**

Your solutions to the assessed coursework may be submitted using this template. Please cut and paste the output from your codes into the correct parts of this file and include your plots and responses to the questions where suggested. Once this template has been completed, you must then create a pdf file for submission. Under Windows or Mac you can use Texmaker + a LaTeX compiler; from the Windows Virtual Desktop this may be accessed as follows:

`Start > UoN Application > (UoN) Texmaker 5`

Open this file under `File`*; to build the pdf file, click the arrow next to* `Quick Build`*; this will then generate the file* `coursework2_submission.pdf`*.*

You may use an alternative document processing system, such as Word, to produce a pdf file containing your results, plots and answers. However, if you do, you must format your answers in the same way as suggested below.

A single zip or tar file containing the file `coursework2_submission.pdf` *and all the files in the requested folders in the checklists below should be submitted on Moodle. Note that all parameters and values should be set within your codes: do NOT use inputs such as those obtained with* `std::cin` *or from the command line.*

James Briant - ID: 14314400

Question 1(c)

File checklist for folder `Q1`**:**

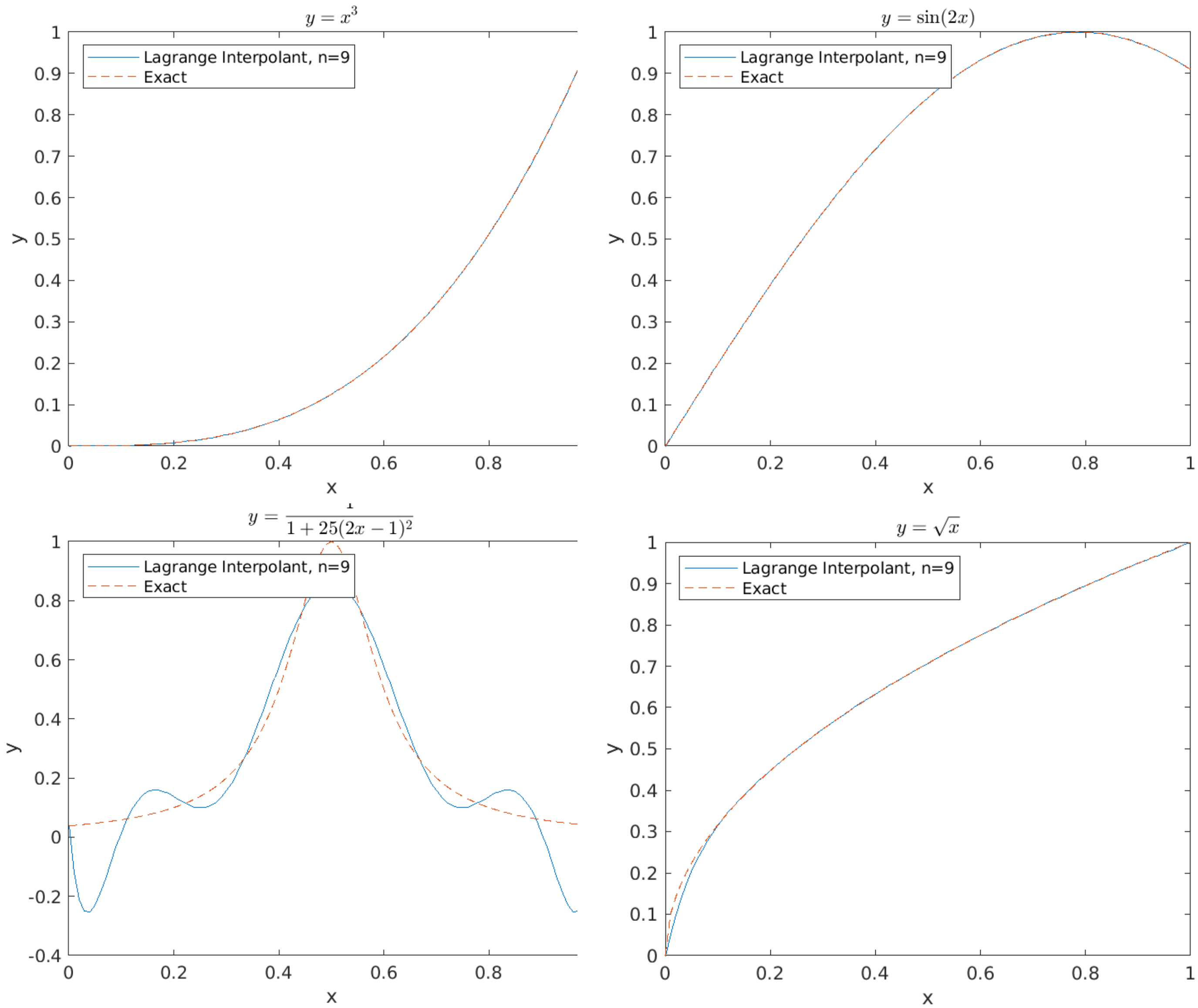
- `AbstractApproximator.cpp`, `AbstractApproximator.hpp`
- `Driver.cpp`
- `Lagrange.cpp`, `Lagrange.hpp`
- `Vector.cpp`, `Vector.hpp`
- For any additional files, provide a `README.txt`

1(c) Enter your output here (max 1 page, display only selected output if necessary).

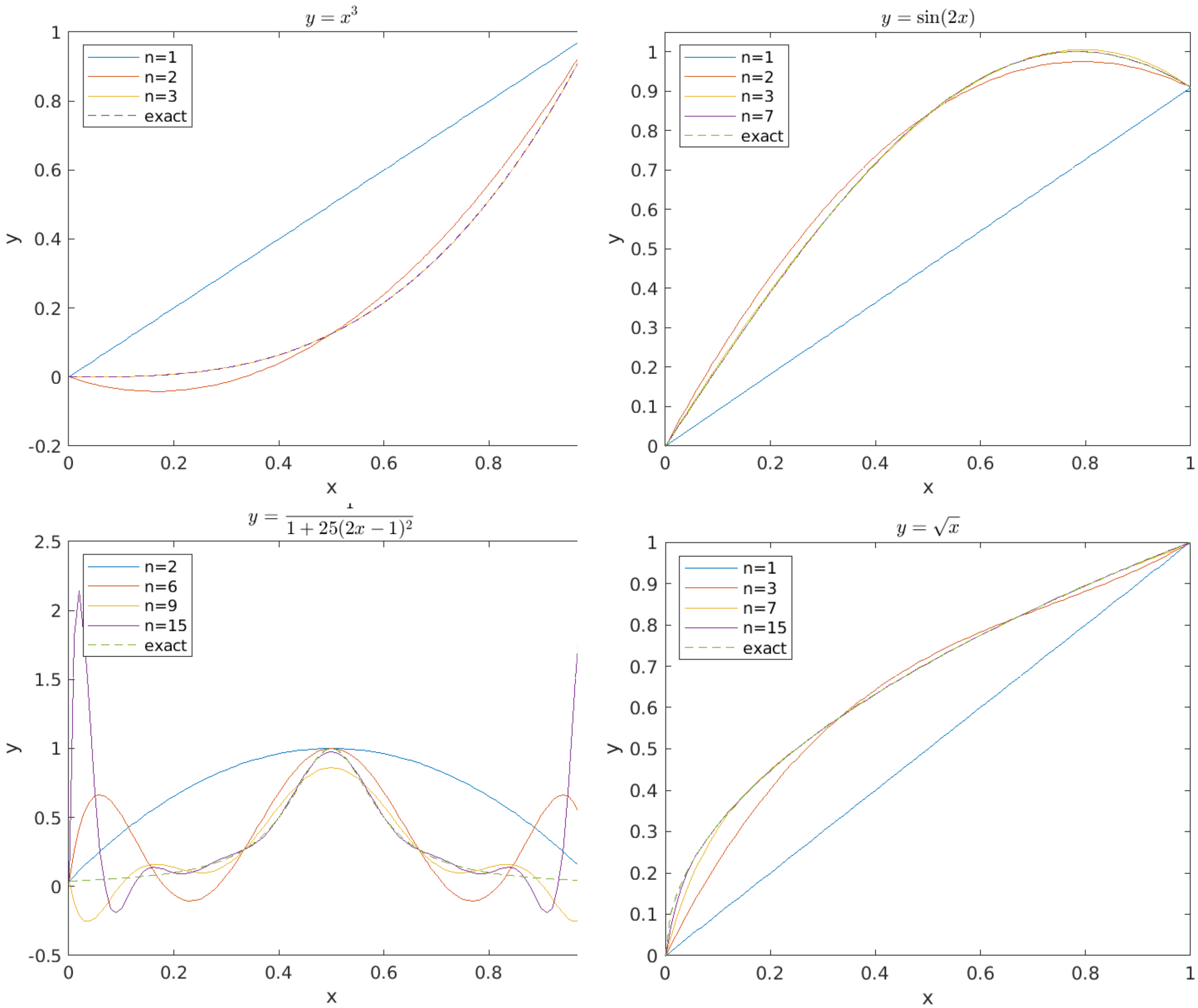
```
Question 1ci

f1: inf-norm approximation = 8.88178e-16
f2: inf-norm approximation = 3.00357e-09
f3: inf-norm approximation = 0.298401
f4: inf-norm approximation = 0.0479874

Question 1cii

f1 approximation:
n = 1: inf-norm approximation = 0.384888
n = 2: inf-norm approximation = 0.048111
n = 3: inf-norm approximation = 2.22045e-16
n = 15: inf-norm approximation = 8.41439e-15

f2 approximation:
n = 1: inf-norm approximation = 0.391094
n = 2: inf-norm approximation = 0.0400074
n = 3: inf-norm approximation = 0.00718133
n = 7: inf-norm approximation = 6.11202e-07

f3 approximation:
n = 2: inf-norm approximation = 0.646154
n = 6: inf-norm approximation = 0.616668
n = 9: inf-norm approximation = 0.298401
n = 15: inf-norm approximation = 2.09903

f4 approximation:
n = 1: inf-norm approximation = 0.25
n = 3: inf-norm approximation = 0.104336
n = 7: inf-norm approximation = 0.0560823
n = 15: inf-norm approximation = 0.0317475
```

1(c) Include your plots and comments here.

The 9^{th} degree polynomials, constructed using 10 points, are displayed below for the functions f_1, f_2, f_3 and f_4.



Polynomials of various degrees are displayed below for the functions f_1, f_2, f_3 and f_4. Here, n denotes the degree of the polynomial (ie. the polynomial is generated using $n+1$ points).



Generally, the higher the degree of polynomial estimate, the better the approximation to the true function. This can clearly be seen in the decreasing infinity-norm estimates for the function f_1, f_2 and f_4. f_3 does not follow this trend. This is known as Runge's phenomenon and occurs because the magnitude of the n-th order derivatives of this particular function grows quickly when n increases. The other functions, and all of their derivatives, are bounded on $[0, 1]$ which avoids the Runge phenomenon.

Since f_1 is a polynomial of degree 3, interpolation of any higher degrees is pointless, as the $n = 3$ degree interpolation is already exact. f_2 and f_4 can be written as Taylor series and so a finite approximation to the Taylor series will provide a reasonable degree of accuracy for moderate n such as $n = 7$.

Question 2(c)

File checklist for folder `Q2`:

- `AbstractQuadratureRule.cpp`, `AbstractQuadratureRule.hpp`
- `Driver.cpp`
- `Gauss4point.cpp`, `Gauss4point.hpp`
- `Matrix.cpp`, `Matrix.hpp`
- `Simpson.cpp`, `Simpson.hpp`
- `Vector.cpp`, `Vector.hpp`
- For any additional files, provide a `README.txt`

2(c) Enter your output here (display only selected output if necessary).

```
Question 2ci - f integral approximations

Simpson's Rule:
f1: 0.25
f2: 0.71253
f3: 0.679487
f4: 0.638071
Gauss-4-Point Rule:
f1: 0.25
f2: 0.708073
f3: 0.185464
f4: 0.667828

Question 2cii - f*L integral approximations

Simpson's Rule
Linear:
   4.16667e-02
   2.08333e-01

Quadratic:
   0.00000e+00
   8.33333e-02
   1.66667e-01


Gauss-4-Point Rule
Linear:
   5.00000e-02
   2.00000e-01

Quadratic:
  -1.66667e-02
```

```
   1.33333e-01
   1.33333e-01
```

Question 2ciii - Li*Lj integral approximations

Simpson's Rule
Linear:

```
   3.33333e-01   1.66667e-01
   1.66667e-01   3.33333e-01
```

Quadratic:

```
   1.66667e-01   0.00000e+00   0.00000e+00
   0.00000e+00   6.66667e-01   0.00000e+00
   0.00000e+00   0.00000e+00   1.66667e-01
```

Gauss-4-Point Rule
Linear:

```
   3.33333e-01   1.66667e-01
   1.66667e-01   3.33333e-01
```

Quadratic:

```
   1.33333e-01   6.66667e-02  -3.33333e-02
   6.66667e-02   5.33333e-01   6.66667e-02
  -3.33333e-02   6.66667e-02   1.33333e-01
```

Question 2civ - Gaussian Elimination

Simpson's Rule
Linear:

```
  -2.50000e-01
   7.50000e-01
```

Quadratic:

```
   0.00000e+00
   1.25000e-01
   1.00000e+00
```

Gauss-4-Point Rule
Linear:

```
  -2.00000e-01
   7.00000e-01
```

Quadratic:

```
   5.00000e-02
   1.25000e-01
   9.50000e-01
```

The estimate for $\boldsymbol{f}$ is exact using the Gauss-4-point algorithm as this provides exact solutions for polynomials up to degree 3, which f_1 is. Simpson's quadrature rule uses 3 points and hence provides a quadratic polynomial approximation. As such, the Simpson's approximations are not as accurate as the Gauss-4-point approximations for $f \times L$ integrals.

For the $L_i \times L_j$ integration approximations required to calculate $\boldsymbol{A}$, Simpson's rule is exact for the linear approximation, but not for the quadratic approximation. Whereas the Gauss-4-point rule is exact for the linear and quadratic approximations. This arises again from the nature of the algorithms and their ability to approximate polynomials of different degrees.

Finally, the Gauss-4-point provides for $\boldsymbol{p}$ the exact solution in both cases whereas Simpson's rule fails to capture all the information required to provide the exact solution. This is due to the inaccuracies in calculating $\boldsymbol{f}$ and $\boldsymbol{A}$ in the previous steps.

Question 3(b)

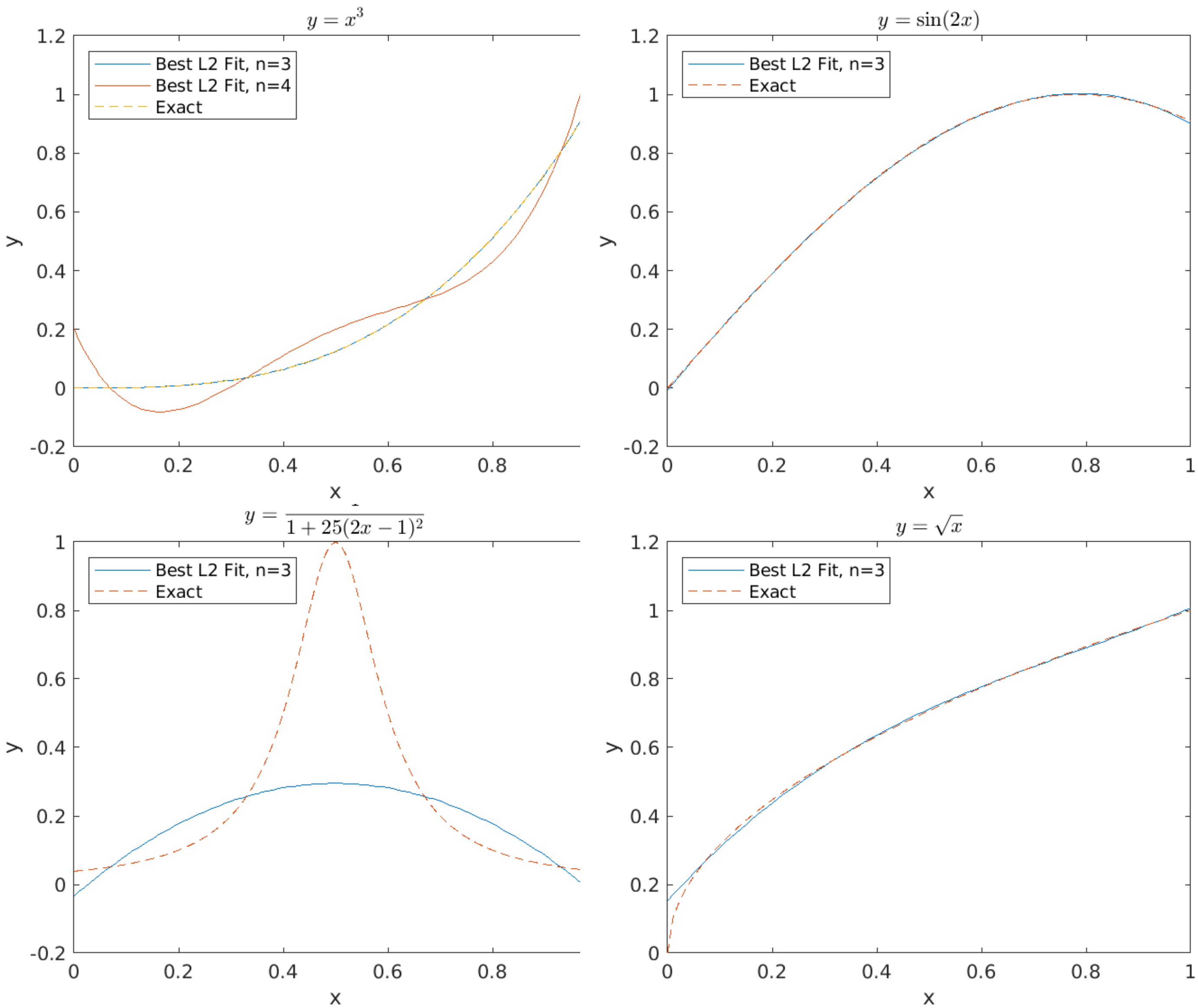
File checklist for folder `Q3`:

- `AbstractApproximator.cpp`, `AbstractApproximator.hpp`
- `AbstractQuadratureRule.cpp`, `AbstractQuadratureRule.hpp`
- `BestL2Fit.cpp`, `BestL2Fit.hpp`
- `Driver.cpp`
- `Gauss4point.cpp`, `Gauss4point.hpp`
- `Matrix.cpp`, `Matrix.hpp`
- `Vector.cpp`, `Vector.hpp`
- For any additional files, provide a `README.txt`

3(b) Enter your output here (display only selected output if necessary).

```
f1
maximum error: 2.22045e-16
2Norm: 5.47973e-17

f2
maximum error: 0.00851443
2Norm: 0.00266734

f3
maximum error: 0.704887
2Norm: 0.23882

f4
maximum error: 0.153048
2Norm: 0.0183238

f1 - Quartic approximation
maximum error: 0.2
2Norm: 0.0694467
```

3(b) Include your plots and comments here.



The polynomial of degree 3 estimates f_1, f_2 and f_4. The cubic L2 norm method does not work well for the f_3. This is reinforced by the large L2-norm and large maximum error of approximately 0.7.

We can also see that the quartic approximation to f_1 does not work well with this method. The best fit method enforces the use of 4 quartic Lagrange polynomials as a basis which is why the cubic polynomial is not approximated well.

Question 4(b)

File checklist for folder Q4:

- AbstractApproximator.cpp, AbstractApproximator.hpp
- AbstractQuadratureRule.cpp, AbstractQuadratureRule.hpp
- Driver.cpp
- Gauss4point.cpp, Gauss4point.hpp
- LocalBestL2Fit.cpp, LocalBestL2Fit.hpp
- Matrix.cpp, Matrix.hpp
- Vector.cpp, Vector.hpp
- For any additional files, provide a README.txt

4(b) Enter your output here (display only selected output if necessary).

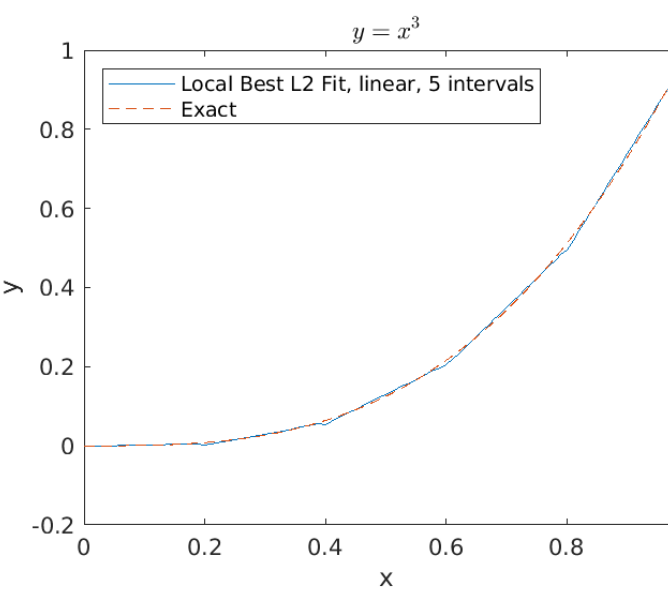
```
Question 4bi
f1, 5 intervals, linear approximation
maximum error: 0.0184
2Norm: 0.00538618

f2, 5 intervals, linear approximation
maximum error: 0.0130564
2Norm: 0.00474825

f3, 5 intervals, linear approximation
maximum error: 0.284314
2Norm: 0.079169

f4, 5 intervals, linear approximation
maximum error: 0.121467
2Norm: 0.0140787

Question 4bii
Linear Approximation, 2 intervals:
f1 - maximum error: 0.1, 2Norm: 0.0325559
f2 - maximum error: 0.0816436, 2Norm: 0.029648
f3 - maximum error: 0.287374, 2Norm: 0.137371
f4 - maximum error: 0.192056, 2Norm: 0.028279

Linear Approximation, 4 intervals:
f1 - maximum error: 0.028125, 2Norm: 0.00845119
f2 - maximum error: 0.020555, 2Norm: 0.00745211
f3 - maximum error: 0.13887, 2Norm: 0.0467014
f4 - maximum error: 0.135804, 2Norm: 0.0164819

Linear Approximation, 8 intervals:
```

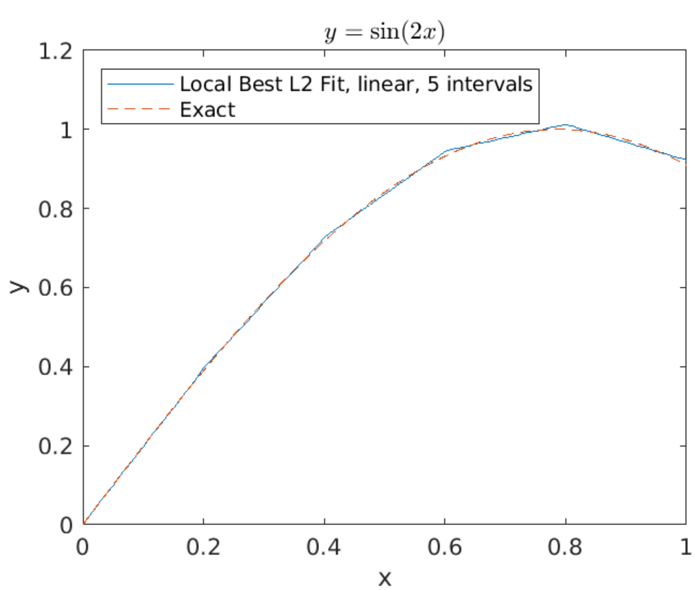
```
f1 - maximum error: 0.007422, 2Norm: 0.0021081
f2 - maximum error: 0.00519501, 2Norm: 0.00184794
f3 - maximum error: 0.0606, 2Norm: 0.0109066
f4 - maximum error: 0.096028, 2Norm: 0.0104574

Cubic Approximation, 2 intervals:
f1 - maximum error: 2.22045e-16, 2Norm: 5.46126e-17
f2 - maximum error: 0.000588427, 2Norm: 0.000164499
f3 - maximum error: 0.17177, 2Norm: 0.0252179
f4 - maximum error: 0.108222, 2Norm: 0.0115399

Cubic Approximation, 4 intervals:
f1 - maximum error: 2.22045e-16, 2Norm: 5.46097e-17
f2 - maximum error: 3.69866e-05, 2Norm: 1.05823e-05
f3 - maximum error: 0.08556, 2Norm: 0.0133053
f4 - maximum error: 0.0765242, 2Norm: 0.00775632

Cubic Approximation, 8 intervals:
f1 - maximum error: 2.22045e-16, 2Norm: 5.46096e-17
f2 - maximum error: 2.42683e-06, 2Norm: 7.00906e-07
f3 - maximum error: 0.0072, 2Norm: 0.00168223
f4 - maximum error: 0.0541108, 2Norm: 0.00541437
```

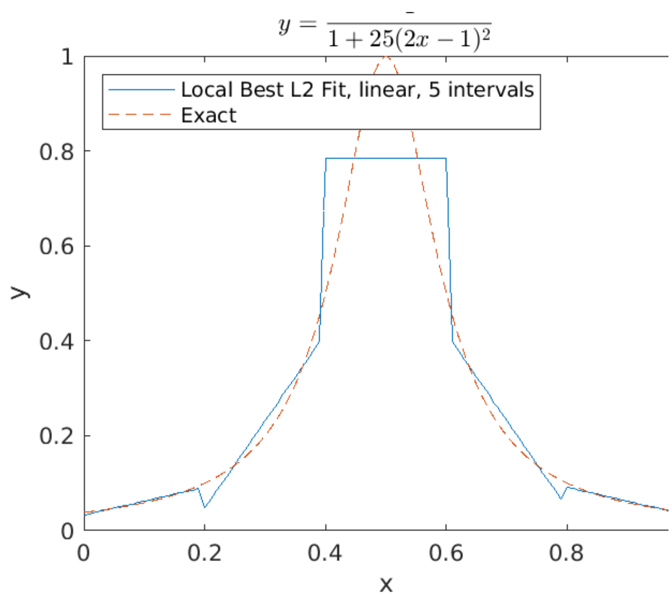
4(b) Include your plots and comments here.

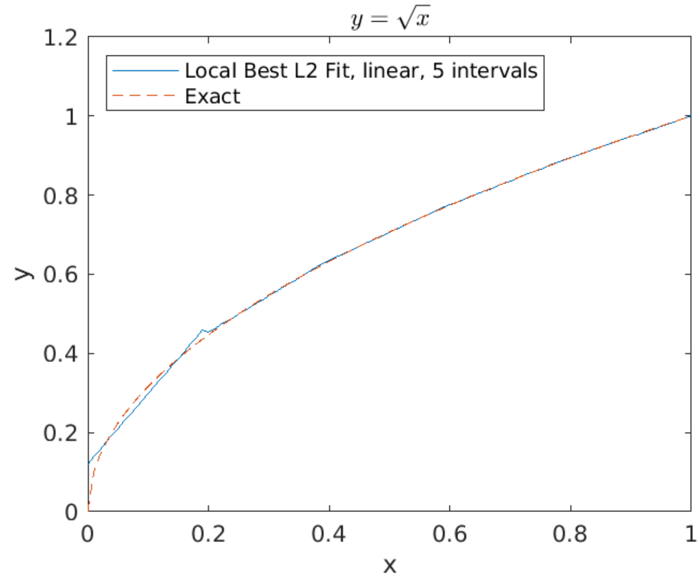
Question 4bi

The 'joltyness' in all of these graphs is caused by MATLAB filling in the discontinuous function approximations. In some cases, the nature of the algorithms and the distinct subintervals are visually obvious, such as f_3 using 5 subintervals with linear approximations.



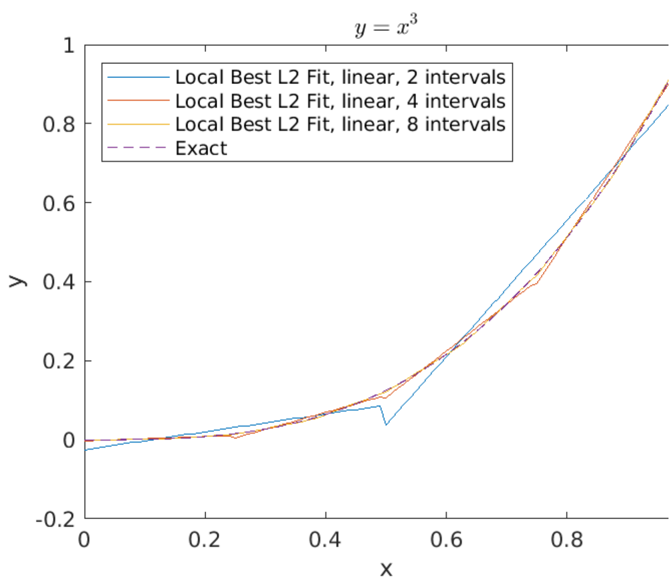


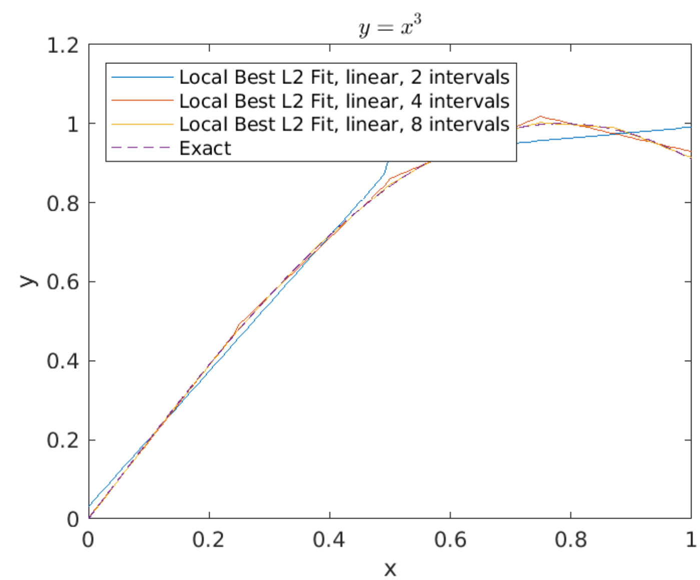


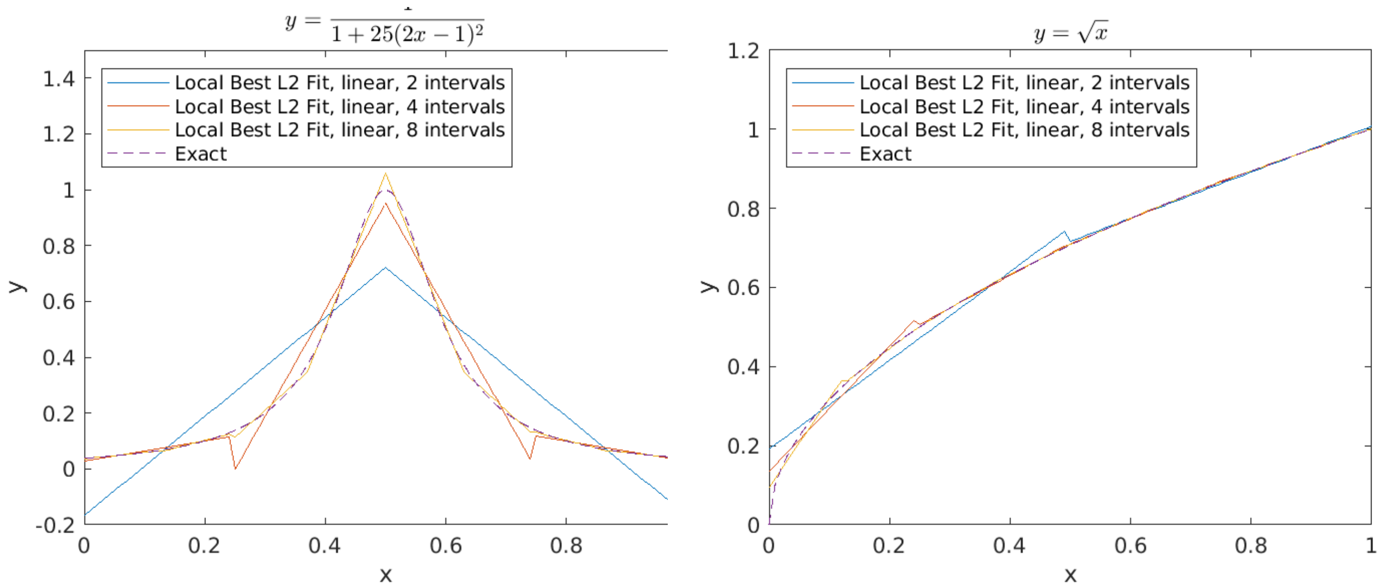


Question 4bii

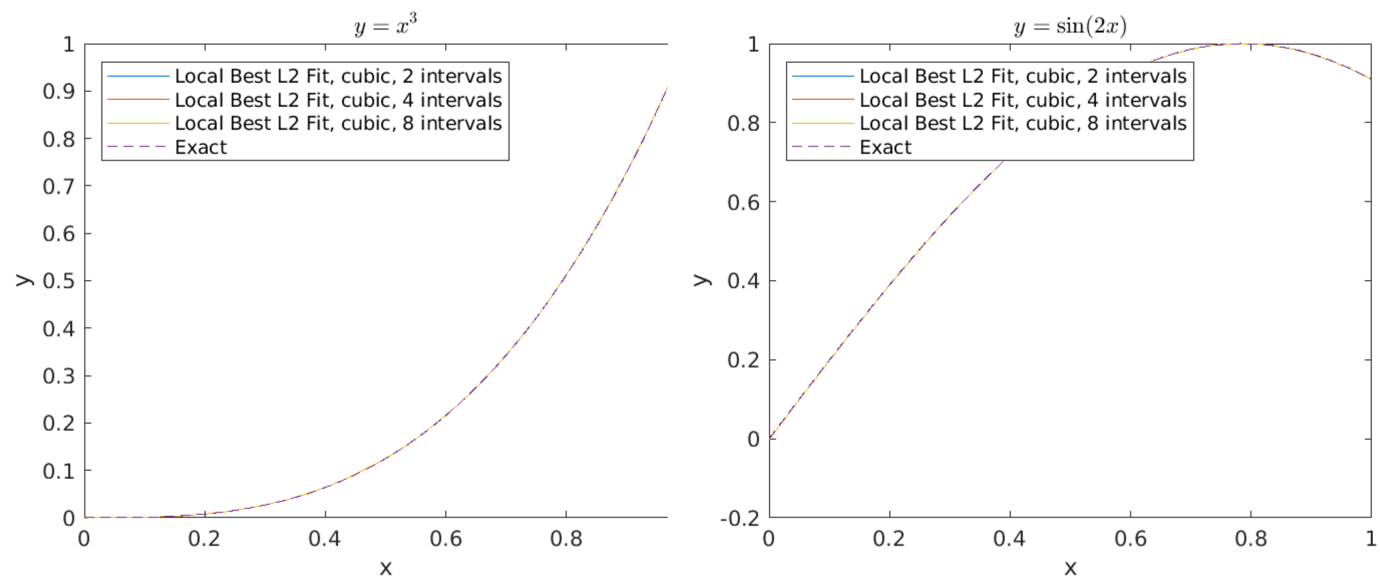
The following 4 plots are the linear approximations to the functions f_1-f_4 using 2, 4 and 8 intervals.

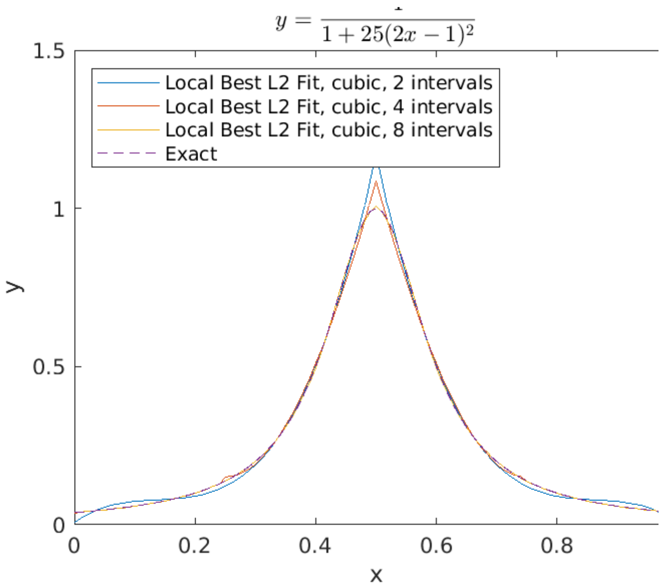


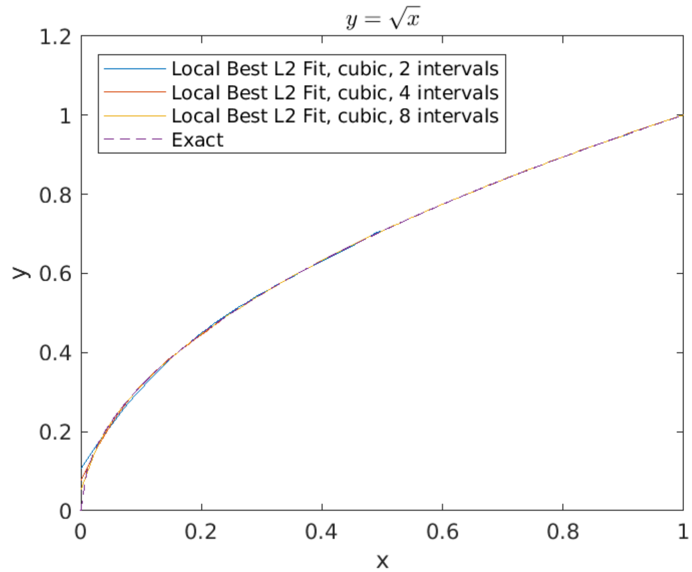




The following 4 plots are the cubic approximations to the functions f_1-f_4 using 2, 4 and 8 intervals.







All of these approximations are visually very similar to the functions being approximated. The approximation for f_1 are of course all exact up to computational rounding as shown by the approximately 0 errors and 2-norm of the errors. The cubic f_2 and f_4 approximations are also very accurate. The biggest improvement over is in f_3 where the cubic approximation are much more accurate than the linear counterparts. Even using 8 subintervals, the 2-norm decreases by almost a factor of 10 between the linear and cubic approximations from 0.0109066 to 0.00168223.

In all cases, the errors for the cubic approximations are better than the linear approximations using the same number of intervals. And in all cases, using more intervals decreases the approximation's error.

Some additional features can be picked out by analysing the errors. Considering the 2-norm for the cubic approximations to f_4, we have the following relationship:

$$\frac{\text{2norm using 8 intervals}}{\text{2norm using 4 intervals}} = \frac{0.00541437}{0.00775632} = 0.698059$$

$$\frac{\text{2norm using 4 intervals}}{\text{2norm using 2 intervals}} = \frac{0.00775632}{0.0115399} = 0.672131$$

$$0.698059 \approx 0.672131$$

This indicates a logarithmic relationship between the error and the number of intervals. Similar relationships also occur in the linear and cubic approximations to f_2 and the linear approximations in f_1, f_3 and f_4. Cubic approximations to f_3 do not follow this pattern.