HW2

Gavin DeBrun

1.

$f_{j-1} = f_j - hf_j + \frac{h^2}{2}f_j'' - \frac{h^3}{6}f_j''' + ...$

$f_{j-2} = f_j - 2hf_j + 2h^2 f_j'' - \frac{4}{3}h^3 f_j''' + ...$

$f_{j-2} - 2f_{j-1} = -f_j + h^2 f_j'' - h^3 f_j''' + ...$

$f_j'' = \frac{f_j - 2f_{j-1} + f_{j-2}}{h^2} + hf_j''' + ...$

Thus, $\tau = hf_j'''$ and so this is an order 1 approximation.

2.

	f_j	f_j'	f_j''	f_j'''	f_j''''
f_j'	0	1	0	0	0
$a_{-1}f_{j-1}$	a_{-1}	$-a_{-1}h$	$a_{-1}\frac{h^2}{2}$	$-a_{-1}\frac{h^3}{6}$	$a_{-1}\frac{h^4}{24}$
$a_0 f_j$	a_0	0	0	0	0
$a_1 f_{j+1}$	a_1	$a_0 h$	$a_0\frac{h^2}{2}$	$a_1\frac{h^3}{6}$	$a_1\frac{h^4}{24}$
$a_2 f_{j+2}$	a_2	$2a_2 h$	$2a_2 h^2$	$\frac{4}{3}a_2 h^3$	$\frac{2}{3}a_2 h^4$

$f_j' + \sum_{k=-1}^{2} a_k f_{j+k} = (a_{-1} + a_0 + a_1 + a_2)f_j + (1 - a_{-1}h + a_1 h + 2a_2 h)f_j' + (a_{-1}\frac{h_2}{2} + a_1\frac{h^2}{2} + 2a_2 h^2)f_j''$
$+ (-a_{-1}\frac{h^3}{6} + a_1\frac{h^3}{6} + \frac{4}{3}a_2 h^3)f_j''' + (a_{-1}\frac{h^4}{24} + a_1\frac{h^4}{24} + \frac{2}{3}a_2 h^4)f_j'''' + ...$

We can set the first four coefficients to 0 to get the following system of equations:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ -h & 0 & h & 2h \\ \frac{h^2}{2} & 0 & \frac{h^2}{2} & 2h^2 \\ -\frac{h^3}{6} & 0 & \frac{h^3}{6} & \frac{4}{3}h^3 \end{bmatrix} \begin{bmatrix} a_{-1} \\ a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

And solving for each a_i gives us the following:

```python
import sympy as sym

h = sym.Symbol("h")
a0 = sym.Symbol("a0")
a1 = sym.Symbol("a1")
a2 = sym.Symbol("a2")
a3 = sym.Symbol("a3")

sym.solve(
    [
        a0 + a1 + a2 + a3,
        -a0 * h + h * a2 + 2 * h * a3 + 1,
        h**2 / 2 * a0 + h**2 / 2 * a2 + 2 * h * a3,
        -(h**3) / 6 * a0 + h**3 / 6 * a2 + 4 / 3 * h**3 * a3,
    ],
    [a0, a1, a2, a3],
)
```

```
{a0: 0.333333333333333*(2.0*h - 1.0)/h**2,
 a1: 0.166666666666667*(4.0 - h)/h**2,
 a2: 0.333333333333333*(-2.0*h - 1.0)/h**2,
 a3: 0.166666666666667/h}
```

$a_{-1} = \frac{1}{3h^2}(2h-1)$

$a_0 = \frac{1}{6h^2}(4-h)$

$a_1 = \frac{1}{3h^2}(-2h-1)$

$a_2 = \frac{1}{6h}$

Now using $f'_j = -\sum_{k=-1}^{2} a_k f_{j+k} + O(h^4)$, we obtain:

$f'_j = \frac{1}{3h^2}(1-2h)f_{j-1} + \frac{1}{6h^2}(h-4)f_j + \frac{1}{3h^2}(2h+1)f_{j+1} - \frac{1}{6h} + O(h^4)$

And now solving for τ:

$\tau = \frac{1}{3h^2}(2h-1)\frac{h^4}{24} + \frac{1}{3h^2}(-2h-1)\frac{h^4}{24} + \frac{2}{3} \cdot \frac{1}{6h}h^4 = -\frac{h^2}{36} + \frac{h^2}{9}$

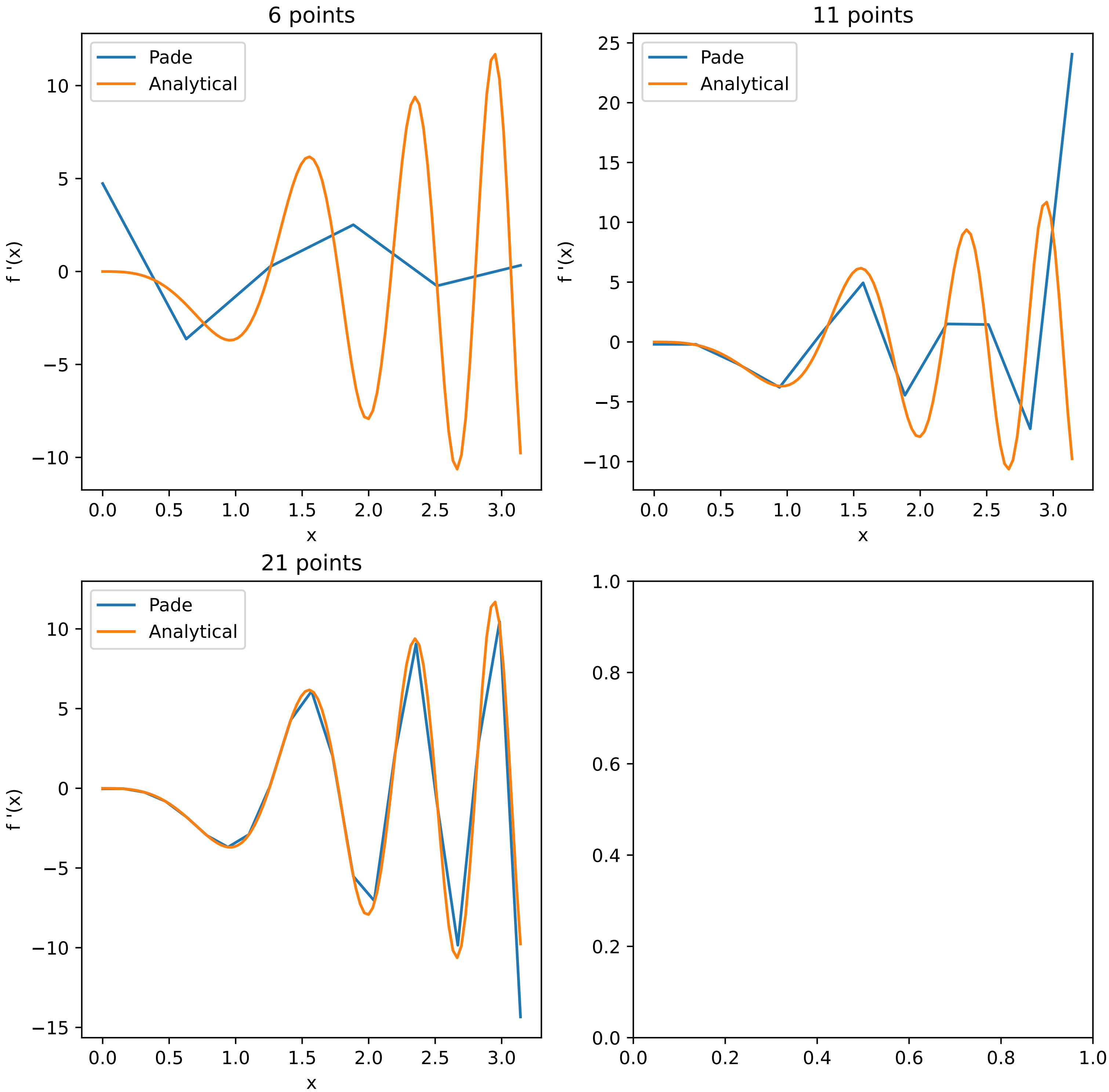
Thus, this is an order 2 approximation.

3.

```python
import numpy as np

def pade(x, f):

    n = f.shape[0]
    h = x[1] - x[0]
    a1 = np.ones(n - 1)
    a1[-1] = 2
    a1 = np.diag(a1, k=-1)
    a2 = 4 * np.ones(n)
    a2[0] = 1
    a2[-1] = 1
    a2 = np.diag(a2)
    a3 = np.ones(n - 1)
    a3[0] = 2
    a3 = np.diag(a3, k=1)
    A = a1 + a2 + a3

    b = np.zeros(n)

    for i, bi in enumerate(b):
        if i == 0:
            b[i] = -2.5 * f[i] + 2 * f[i + 1] + 0.5 * f[i + 2]

        elif i == n - 1:
            b[i] = 2.5 * f[i] - 2 * f[i - 1] - 0.5 * f[i - 2]

        else:
            b[i] = 3 * (f[i + 1] - f[i - 1])
    b /= h

    return np.linalg.solve(A, b)
```

4a.

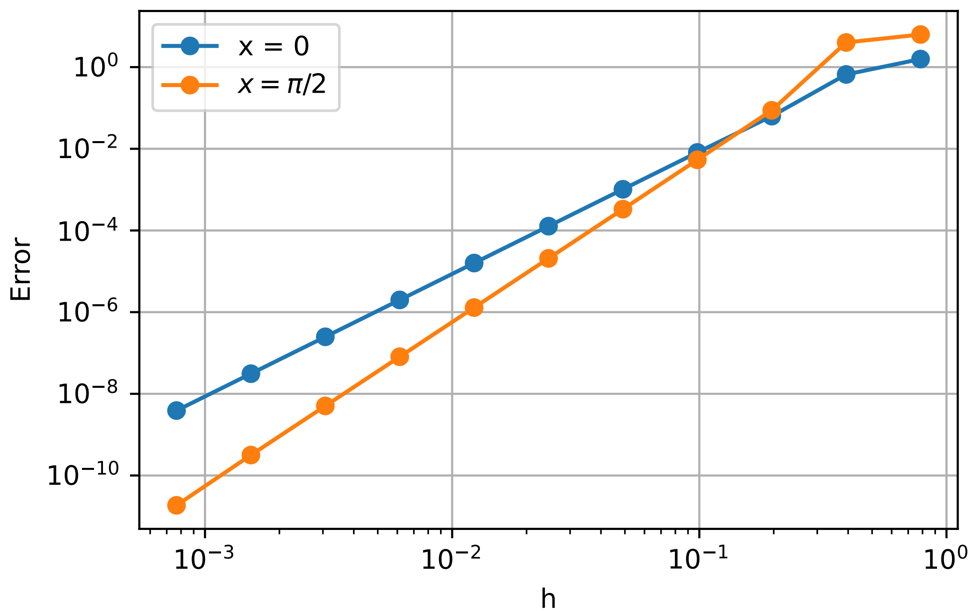
```
import matplotlib.pyplot as plt
def f(x):
    return np.cos(2 * x**2)


def fp(x):
    return -4 * x * np.sin(2 * x**2)


x0 = np.linspace(0, np.pi, 100)
fp0 = fp(x0)

x1 = np.linspace(0, np.pi, 6)
x2 = np.linspace(0, np.pi, 11)
x3 = np.linspace(0, np.pi, 21)

f1 = f(x1)
fp1 = pade(x1, f1)

f2 = f(x2)
fp2 = pade(x2, f2)

f3 = f(x3)
fp3 = pade(x3, f3)

fig, ax = plt.subplots(2, 2, figsize=(10, 10))
ax[0, 0].plot(x1, fp1, label="Pade")
ax[0, 0].plot(x0, fp0, label="Analytical")
ax[0, 0].set_title("6 points")
ax[0, 0].legend()
ax[0, 0].set_ylabel("f '(x)")
ax[0, 0].set_xlabel("x")

ax[0, 1].plot(x2, fp2, label="Pade")
ax[0, 1].plot(x0, fp0, label="Analytical")
ax[0, 1].set_title("11 points")
ax[0, 1].legend()
ax[0, 1].set_ylabel("f '(x)")
ax[0, 1].set_xlabel("x")

ax[1, 0].plot(x3, fp3, label="Pade")
ax[1, 0].plot(x0, fp0, label="Analytical")
ax[1, 0].set_title("21 points")
ax[1, 0].legend()
ax[1, 0].set_ylabel("f '(x)")
ax[1, 0].set_xlabel("x")

plt.show()
```



4b.

```
n = np.arange(2.0, 13.0, 1)
h = np.pi / 2 ** (n)
err0 = np.zeros(n.shape[0])
err_pi_2 = np.zeros(n.shape[0])

for i, hi in enumerate(h):
    grid = np.linspace(0, np.pi, int(np.pi / hi) + 1)
    fpade = pade(grid, f(grid))
    fp0 = fp(grid)
```

```
    err0[i] = np.abs(fpade[0] - fp0[0])
    ni = fpade.shape[0]
    err_pi_2[i] = np.abs(fpade[ni // 2] - fp0[ni // 2])

plt.loglog(h, err0, label="x = 0", marker="o")
plt.loglog(h, err_pi_2, label=r"$x = \pi/2$", marker="o")
plt.xlabel("h")
plt.ylabel("Error")
plt.legend()
plt.grid()
plt.show()

m0, b0 = np.polyfit(np.log(h), np.log(err0), 1)
m_pi_2, b_pi_2 = np.polyfit(np.log(h), np.log(err_pi_2), 1)

print(f"The log-error slope at x = 0: {np.round(m0, 3)}")
print(f"The log-error slope at x = pi/2: {np.round(m_pi_2, 3)}")
```




```
The log-error slope at x = 0: 2.947
The log-error slope at x = pi/2: 3.981
```

The log-error slope at the boundary is approximately 3, which is exactly as expected, as the scheme at the boundary points is $O(h^3)$. The log-error slope at the center is approximately 4, which is also expected as the scheme at the interior points is $O(h^4)$.