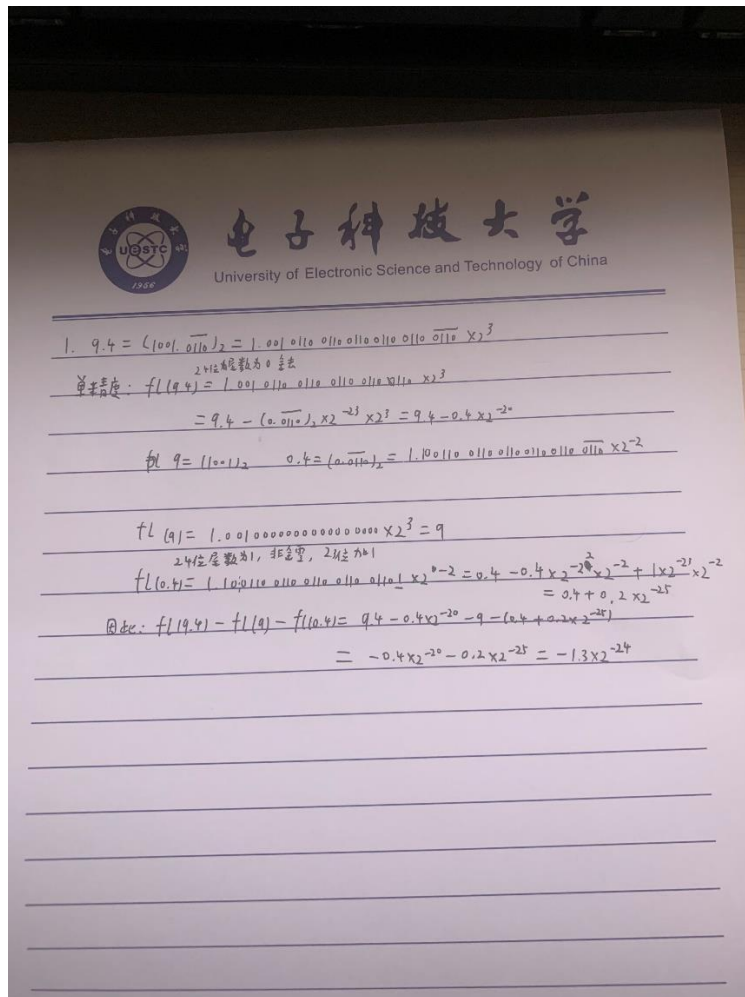


第一次作业

1.分析过程:



计算机实践:

程序

```
器 - C:\Users\91235\Desktop\数值分析\Pro1.m
o2.m x Pro3.m x Pro1.m x +
a=single(9.4);b=single(9);c=single(0.4);
d=a-b-c;
fprintf('%12f',d);
```

运行结果:

```
命令行窗口
>> Pro1
fx -0.000000387430>>
```

2. 优化算法采用秦九韶算法

X=2 时直接算法和优化算法程序:

```
>> x1=2;x2=2.222222;
    PX1=0;PX2=0;PX3=0;PX4=0;
    tic
    for i=1:10^8
        PX1=1+2*x1^3+3*x1^7+4*x1^11+5*x1^15;
    end
    toc
    fprintf('%.12f',PX1)

>> tic
    for i=1:10^8
        PX3=1+x1^3*(2+x1^4*(3+x1^4*(4+5*x1^4)));
    end
    toc
    fprintf('%.12f',PX3)
```

运行结果:

```
    fprintf('%.12f',PX1)
时间已过 57.642915 秒。
172433.000000000000>>
    fprintf('%.12f',PX3)
时间已过 55.091978 秒。
172433.000000000000>>
```

在 Matlab 程序中, 为了让两个算法的时间差异明显, 我选择循环计算 1 亿次, 由运行结果我们可以看出 X=2 时优化算法比直接算法快了 4.4%。

X=2.222222 时直接算法和优化算法程序:

```
] for i=1:10^8
    PX2=1+2*x2^3+3*x2^7+4*x2^11+5*x2^15;%x=2.222222时采用直接算法
- end
toc
fprintf('x=2.222222直接算法结果: %.12f\n',PX2)
tic
] for i=1:10^8
    PX4=1+x2^3*(2+x2^4*(3+x2^4*(4+5*x2^4)));%x=2.222222时采用优化算法
- end
toc
fprintf('x=2.222222优化算法结果: %.12f\n',PX4)
```

运行结果:

```
time = 56.505716
822689.845162201560>>
time = 55.782686
822689.845162201440>>
```

在 Matlab 程序中, 为了让两个算法的时间差异明显, 我选择循环计算 1 亿次, 由运行结果我们可以看出在 X=2.222222 优化算法比直接算法快了 1.3%。

3.计算机实践:

编辑器 - C:\Users\91235\Desktop\数值分析\Pro3.m

```
Pro2.m  Pro3.m  Pro1.m  +
I=0; I0=exp(1)-1;
fprintf(' 正向推导: \n');
fprintf(' I0: %.12f\n', I0);
for i=1:20
    I=exp(1)-i*I0;
    I0=I;
    fprintf(' I%d: %.12f\n', i, I);
end
I1=0; I20=(exp(1)-1)/21;
fprintf(' 反向推导: \n');
fprintf(' I20: %.12f\n', I20);
for i=1:20
    I1=(exp(1)-I20)/(21-i);
    I20=I1;
    fprintf(' I%d: %.12f\n', 20-i, I1);
end
```

运行结果:


正向推导：

I0: 1.718281828459
I1: 1.000000000000
I2: 0.718281828459
I3: 0.563436343082
I4: 0.464536456131
I5: 0.395599547802
I6: 0.344684541647
I7: 0.305490036930
I8: 0.274361533022
I9: 0.249028031257
I10: 0.228001515886
I11: 0.210265153710
I12: 0.195099983941
I13: 0.181982037220
I14: 0.170533307383
I15: 0.160282217715
I16: 0.153766345019
I17: 0.104253963134
I18: 0.841710492049
I19: -13.274217520469
I20: 268.202632237842

反向推导：

I20: 0.081822944212
I19: 0.131822944212
I18: 0.136129414960
I17: 0.143452911861
I16: 0.151460524506
I15: 0.160426331497
I14: 0.170523699797
I13: 0.181982723476
I12: 0.195099931153
I11: 0.210265158109
I10: 0.228001515486
I9: 0.249028031297
I8: 0.274361533018
I7: 0.305490036930
I6: 0.344684541647
I5: 0.395599547802
I4: 0.464536456131
I3: 0.563436343082
I2: 0.718281828459
I1: 1.000000000000
I0: 1.718281828459

两种方式结果分析:

 **电子科技大学**
University of Electronic Science and Technology of China

3. 递推公式: $I_0 = \int_0^1 e^x dx = e - 1$ $I_1 = \int_0^1 x e^x dx = x e^x|_0^1 - \int_0^1 e^x dx = e - I_0$
 $I_2 = \int_0^1 x^2 e^x dx = x^2 e^x|_0^1 - 2 \int_0^1 x e^x dx = e - 2I_1$
观察可得 $\begin{cases} I_0 = e - 1 \\ I_n = e - n I_{n-1} \quad n \geq 1 \end{cases}$

正向推导中: I_{10} 为负数 $\because I_n = \int_0^1 x^n e^x dx \approx \int_0^1 x^n = \frac{1}{n+1}$ 当 $x \rightarrow 0$
 $I_n = \int_0^1 x^n e^x dx \approx e \int_0^1 x^n dx = \frac{e}{n+1}$ 当 $x \rightarrow 1$
 $\therefore I_n > 0$ 因此正向推导出现误差

误差原因分析: $e(I_n) = (-1)^n n! e(I_0)$
随着正向递推的进行, 阶乘数迅速增大, 使得后面的误差被放大的很多, 即使 I_0 误差很小, 也随着阶乘的增大而变得很大。

反向推导: $I_{n-1} = \frac{e - I_n}{n}$, $e(I_{n-1}) = -\frac{1}{n} e(I_n) = -\frac{1}{n!} e(I_0)$
此时误差是逐渐减小, 收敛的
因此反向推导是一种数值稳定的算法