114847849
Kangning Fengwu
AMS326
Final
Source Code Link: https://github.com/kfengwu/AMS326.git

Problem F.1 (7.5 points) We know by some "magical" calculations that

$$I = \iint_{-1}^{1} u(x,y)dxdy \approx 4.028423$$

where the integrand is

$$u(x,y) = e^{(x^6-y^4)}$$

which looks like

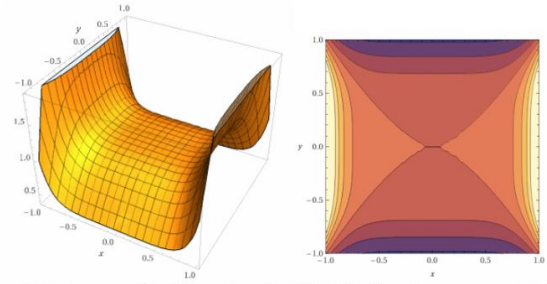


Figure 1. The integrand for this problem: the 3D plot (left) and the contour plot (right).

Please compute the integral by writing programs for two implementations of the Monte Carlo method with N samples and other given parameters/conditions. You need sufficient programming details in your implementations.

Implementation	N	Your Integral	Your Error*	Credits to Give
#1: Using samples drawn randomly and uniformly in $[-1,1]\times[-1,1]$	10^6			3.0
#2: Using samples drawn randomly with density proportional to $\lvert\nabla u(x,y)\rvert$ in $[-1,1]\times[-1,1]$	10^6			4.5

Algorithm Description:

The program estimates the integral of the function u(x,y) = e^(x^6−y^4) over the domain [−1,1]^2 using two Monte Carlo integration techniques: uniform sampling and importance sampling with rejection sampling.

In the uniform sampling method, random points are generated uniformly within the square domain [−1,1]^2. For each point, the function u(x,y) is evaluated, and the average of these function values is computed. Since the area of the domain is 4, the average is scaled by this factor to estimate the integral.

In the importance sampling method, the integral is approximated by sampling from a distribution proportional to the gradient of u(x,y). This is done through rejection sampling, where candidate points are generated randomly, and only those for which the gradient is above a randomly chosen threshold are accepted. After collecting enough samples, the function values are weighted by the inverse of the sampling probability to account for the biased sampling. This results in a more efficient estimate of the integral, particularly for functions with high variance.

Result:

```
Uniform Sampling Estimate:    4.03042417, Error: 2.00116560e-03
Importance Sampling Estimate: 4.07293324, Error: 4.45102381e-02
```

Problem F.2 (7.5 points) Please write program(s) to carry out the following tasks:

(1) (0.5 point) To generate matrices $A^{n\times n}$ and $B^{n\times n}$ with elements $a_{ij}, b_{ij} \sim U(-1, 1)$ where $n = 2^{10}$.
(2) (4.0 point) To compute $C = A \times B$ by Strassen algorithm of 2 levels.
(3) (3.0 point) To compute C^{-1} by any algorithm.

Algorithm Description:

This program generates two random square matrices, A and B, of size n x n where n=2^10, with elements drawn uniformly from the range [-1, 1]. The core computation of the program involves using a recursive, two-level implementation of Strassen's algorithm to multiply matrices A and B to produce matrix C. Strassen's algorithm optimizes matrix multiplication by breaking down the matrices into smaller submatrices and using only seven multiplications instead of the conventional eight. After computing the product matrix C, the program attempts to calculate its inverse. If the matrix C is invertible, it computes and prints the inverse, otherwise, it handles the exception and reports that the matrix is singular and cannot be inverted.

Result:

The program outputs the shape, min, max, mean and standard deviation of the matrices, and it only prints out the top-left 5x5 block of the matrices due to large size.

```
Matrix A Summary:
Shape: (1024, 1024)
Min: -1.0000, Max: 1.0000
Mean: 0.0006, Std: 0.5774
Top-left 5x5 block:
[[-0.76870345  0.85997965 -0.95300084  0.07317817 -0.00312179]
 [-0.93527582 -0.89196418 -0.4150222  -0.64382729  0.76970138]
 [ 0.69024808 -0.18594639 -0.59029297  0.91589595  0.02028845]
 [-0.19075191 -0.41059809 -0.16543797  0.95388206 -0.9698617 ]
 [-0.87254934  0.61164596  0.71493096  0.16867962 -0.96468309]]

Matrix B Summary:
Shape: (1024, 1024)
Min: -1.0000, Max: 1.0000
Mean: -0.0001, Std: 0.5775
Top-left 5x5 block:
[[ 0.81654125 -0.05588382  0.0193575   0.89632464 -0.66726928]
 [ 0.91694933 -0.70212972  0.35788554 -0.6274952  -0.65854974]
 [ 0.90515829  0.38037161  0.22226894 -0.173766   -0.00570392]
 [-0.19031285 -0.25999505  0.53600127 -0.80662835 -0.27402138]
 [-0.06644189 -0.10464241  0.4922487  -0.98511817  0.43239957]]

Matrix C = A x B Summary:
Shape: (1024, 1024)
Min: -53.4480, Max: 49.1750
Mean: -0.0070, Std: 10.6736
Top-left 5x5 block:
[[ -8.68563503 -12.98958381  -2.46016339  11.97220496 -11.
51635693]
 [ -4.20162595  -1.51527331 -14.25533296  12.03809847  14.
 69867424]
 [ -5.55193909   0.11184008  10.06455733   5.79333069  -1.
 73298269]
 [-16.24240937  -6.48828515   4.16025518 -13.96762937   6.
 15164313]
 [-20.14395059  26.62632719   0.72491329  -9.6803652  -14.
 70445206]]

Matrix C^-1 Summary:
Shape: (1024, 1024)
Min: -1.3808, Max: 1.2406
Mean: -0.0000, Std: 0.1381
Top-left 5x5 block:
[[-0.04755056  0.06234923  0.08046703  0.0663871  -0.01879132]
 [-0.01274655  0.09257443 -0.014114    0.02692695 -0.14833713]
 [-0.09635931  0.15673643  0.17969778  0.12377707 -0.11348946]
 [-0.01697882 -0.05373948  0.04590336 -0.04588159  0.1018524 ]
```

Problem F.3 (7.5 points) Please solve the following IVP numerically, and complete other relevant tasks, by programming any algorithm(s) of your choice.

(1) (2.5 points) Select $N = 10^4$ uniform mesh points for $t \in [0,5]$ to find the numerical solution $x(t)$ of

$$\begin{cases} \exp(-x') = x' + x^3 - 3\exp(-t^3) \\ x(t=0) = 1 \end{cases}$$

For this part, you do not need to tabulate N pairs of (t, x) values, but you must plot the solution.

(2) (2.5 points) Take 6 points from your solution above: $x(t=0), x(t=1), \ldots, x(t=5)$ and interpolate them by a polynomial $P_5(t)$. Plot $P_5(t)$ and report the 6 coefficients.

(3) (2.5 points) Fit the above 6 points by the t-exp fit to a function with fitting parameters α and β. Plot $x_{\text{Fit}}(t)$ and report the values of α and β.

$$x_{\text{Fit}}(t) = 1 + \alpha t * e^{-\beta t}$$

Note: I deliberately included "-1" to make fitting easier and $x(t) - 1$ may look like:

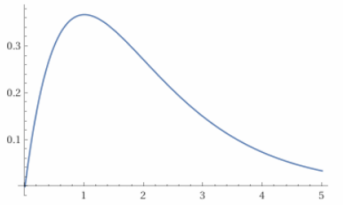
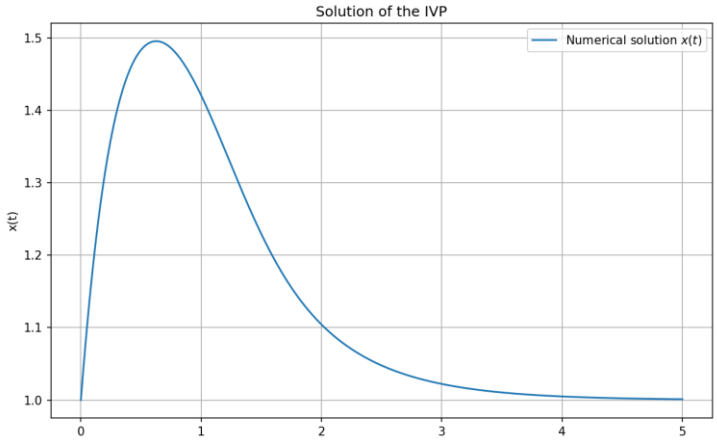


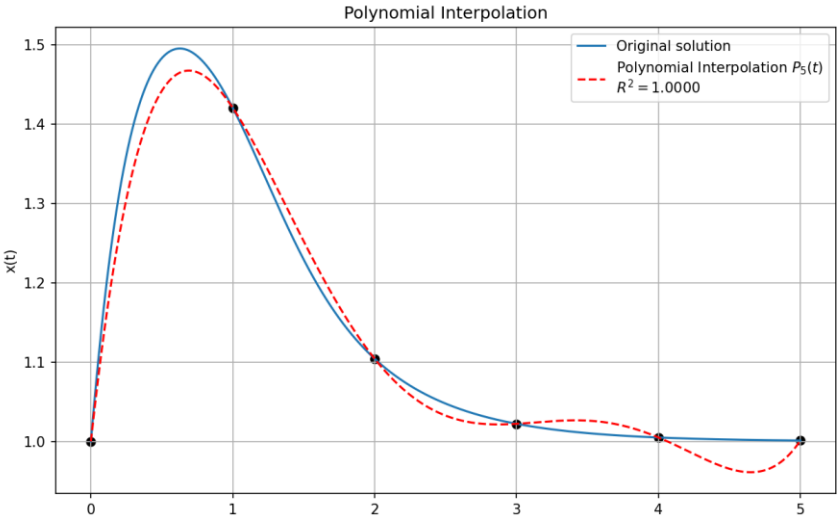
Figure 2. The likely shape of the solution for this problem.

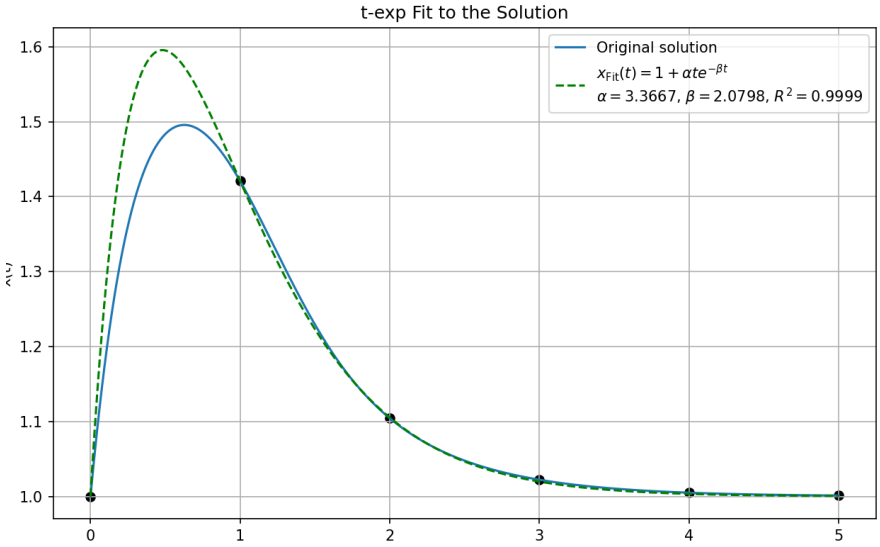
Algorithm Description:

This program tackles three key numerical analysis tasks. First, it solves an initial value problem defined by a nonlinear differential equation using the implicit Euler method, iteratively applying the Newton-Raphson method for root-finding at each time step to compute the solution over a given time range. The resulting solution is plotted against time. In the second part, the program performs polynomial interpolation on six sample points taken from the IVP solution using a degree-5 polynomial, calculating the residuals and R^2 to evaluate the fit quality, and visualizing the polynomial curve alongside the original data. Lastly, the program attempts to fit the solution to a custom model $x_{\text{Fit}}(t) = 1 + \alpha t * e^{-\beta t}$ by performing a grid search over possible values of the parameters alpha and beta, selecting the values that minimize the least-squares error. The fitted curve is plotted, and the optimal parameters, along with the R^2 value, are reported. Each part is visually presented using matplotlib for clarity, showcasing numerical solution techniques, interpolation, and curve fitting.

Result:








```
Polynomial Coefficients (highest degree to constant):
[ 0.01049775 -0.15253357  0.81470082 -1.90243607  1.65047703
1.        ]
Fitted Parameters:
 alpha = 3.3667
 beta = 2.0798
```