Assignment 4

1 Problems

Problem 1 (30 marks). *(Example of curve-fitting).* Consider a second order ODE for $u = u(t)$, for $0 \leq t \leq T$,

$$m\frac{\mathrm{d}^2u}{\mathrm{d}t^2} + c\frac{\mathrm{d}u}{\mathrm{d}t} + ku = 0 \tag{1}$$

with initial conditions

$$u(0) = u_0, \qquad \frac{\mathrm{d}u}{\mathrm{d}t}(0) = \dot{u}_0. \tag{2}$$

Consider $T = 10, m = 1, c = 0.05, k = 0.75, u_0 = 0, \dot{u}_0 = 1$ and take $\Delta t = 0.001$ or smaller for numerical method. Above ODE is frequently encountered when modeling a spring-dashpot system. Specifically, m is the mass attached to the spring, k is the stiffness of spring, and c is the damping coefficient of the dashpot.

(i) ***Implement*** a numerical method to solve (1) numerically using both methods described in the hints. ***Plot*** the solutions from both methods.

(ii) We want to develop a simple model of (1) using curve-fitting such that for a given stiffness k, we can predict the displacement $u(T)$ at final time T. We also want to take into account the uncertainty in the initial condition in developing our predictive model.

Consider different values of stiffness: $k_1 = k, k_2 = k + \Delta k, k_3 = k + 2\Delta k, ..., k_{N_k+1} = k + N_k\Delta k$ with $k = 0.75$, $\Delta k = 0.01$, and $N_k = 50$ so the lower and upper value of k are 0.75 and 1.25, respectively. Also, let initial condition is probabilistic and given by $u_0 = \mathcal{N}(0, \sigma)$, where $\mathcal{N}(\mu, \sigma)$ is the Gaussian probability distribution function with mean μ and standard deviation σ. Let $\sigma = 0.1$.

For each $i = 1, 2, .., N_k + 1$, let k_i is the stiffness of spring. Consider 10 samples of initial condition u_0 from Gaussian distribution $\mathcal{N}(0, \sigma)$. Using u_0 and k_i, ***solve*** (1) and ***record*** value of $u(T)$ for k_i.

Provide table where the first column is different stiffness k_i, $i = 1, 2, ...$, and the second column is $u(T)$ using the stiffness k_i. Clearly, we will have 10 different values of $u(T)$ for each k_i because we consider 10 different samples of initial condition u_0.

(iii) ***Plot*** the tabular data in Matlab where stiffness and displacement are in the x and y-axis, respectively. From the plot, ***explain*** what type of curve-fitting (regression or interpolation) is preferred to obtain a function $f = f(k)$ which gives displacement $u(T)$ for a given k?

(iv) ***Perform*** curve-fitting using the method selected in (iii). For the fitting curve, you can choose either polynomial or sinusoidal basis functions. Try different number of basis functions to see you have a good curve fit.

Plot the fitted curve in the same plot where you have plotted the data in (iii).

(v) ***Provide*** prediction of $u(T)$ using fitted curve in (iv) as well as the actual value of $u(T)$ by solving the ODE (1) with $u_0 = 0$ for the following values of k:

$$0.65, 0.71, 0.83, 0.96, 1.02, 1.09, 1.17, 1.26, 1.34.$$

Provide the values in the table where the first column is stiffness k, second column is $u(T)$ predicted from the fitted curve, third column is $u(T)$ by solving (1) with $u_0 = 0$, and the last column is the error in predicted value and computed value of $u(T)$.

Problem 2 (20 marks). *(Example of linear regression using nonlinear basis functions).* Consider three chemicals in seawater. Each chemical decays at different rates, say, $1.5, 0.3, 0.05$, respectively, in seawater. The total amount of chemical at a given time t is the sum of the three chemicals:

$$u(t) = A\exp[-1.5t] + B\exp[-0.3t] + C\exp[-0.05t], \tag{3}$$

where A, B, C is the amount of the three chemicals initially.

Using the following data

$$\begin{bmatrix} t & 0.5 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 9 \\ u(t) & 6 & 4.4 & 3.2 & 2.7 & 2 & 1.9 & 1.7 & 1.4 & 1.1 \end{bmatrix}$$

compute values of A, B, C using the linear regression (least-squared linear regression method).

Problem 3 (20 marks). *(Polynomial interpolation).* ***Implement*** Matlab code that given n number of paired data (x_i, y_i), computes $(n-1)$-th order polynomial using the three methods we have learned in the class: *direct polynomial interpolation*, *Newton's polynomial interpolation*, and *Lagrange's polynomial interpolation*.

Particularly, ***use*** your polynomial interpolation codes to fit a second order polynomial to the following three paired data:

$$(-2, 4), \quad (0, 2), \quad (2, 8).$$

Provide values of coefficients in the three methods and ***plot*** the resulting polynomial.

Problem 4 (20 marks). *(Runge's phenomenon).* ***Interpolate*** the function

$$f(x) = \frac{1}{1 + 25x^2} \tag{4}$$

with a tenth-order polynomial in the interval $[-1, 1]$ using

(a) eleven equally spaced points,

(b) and the so called *Chebysev points* $x_i = \cos\left(\frac{2i-1}{11}\frac{\pi}{2}\right), \quad i = 1, \ldots, 11$.

Use the Matlab built-in functions `polyfit` and `polyval` (more details in hints). ***Plot*** the results together with the original function in the same figure.

Problem 5 (10 marks). *(Gauss quadrature integration method).* Recall that Trapezoidal and Simpson's 1/3rd and 3/8th (or, generally, methods based on piecewise interpolation) rules for approximation of integral

$$I[f] = \int_a^b f(x)\mathrm{d}x \tag{5}$$

are applied when the values of function $f(x_i)$ is provided at pre-specified discrete points x_i (often uniformly-spaced points). On the other hand, if we had a flexibility of choosing points x_i, we can have a method that uses few points and is more accurate than the previous methods. Gauss quadrature (also called Gauss-Legendre) method and Richardson's extrapolation are two such methods.

Consider $a = 0, b = 3$ and $f = f(x) = xe^{1.5x}$. ***Compute*** the integration $I[f]$ exactly. Also, ***compute*** the approximation of $I[f]$ denoting $\hat{I}[f]$ using two and three-point Gauss quadrature method. For both two and three points method, ***compute*** the true percentage error $100 * (I[f] - \hat{I}[f])/I[f]$.

2 Hints

1. Solving second order ODE. Notice that we can convert the second order ODE

$$m\frac{\mathrm{d}^2u}{\mathrm{d}t^2} + c\frac{\mathrm{d}u}{\mathrm{d}t} + ku = 0$$

into two first order ODEs for displacement $u = u(t)$ and velocity $\mathrm{d}u/\mathrm{d}t = v(t)$ as follows

$$\begin{aligned}\frac{\mathrm{d}u}{\mathrm{d}t} &= v,\\ \frac{\mathrm{d}v}{\mathrm{d}t} &= f(u, v),\end{aligned} \tag{6}$$

where $f(u, v) = -\frac{c}{m}v - \frac{k}{m}u$. We also have two initial conditions

$$u(0) = u_0, \qquad v(0) = \dot{u}_0.$$

Let $t_1 = 0, t_2 = \Delta t, t_3 = 2\Delta t, ..., t_{N_t+1} = N_t\Delta t$ are discrete times. There are several methods to solve the above two coupled first order ODEs. Here we list two which are used very frequently:

1 *Forward Euler method.* Applying forward difference approximation to (6), we get following numerical method, for $i = 1, 2, .., N_t$,

$$\begin{aligned}u(t_{i+1}) &= u(t_i) + \Delta t v(t_i),\\ v(t_{i+1}) &= v(t_i) + \Delta t f(u(t_i), v(t_i)),\end{aligned} \tag{7}$$

where $f(u, v) = -\frac{c}{m}v - \frac{k}{m}u$.

2 *Velocity-Verlet method.* We first compute velocity at mid point $t_{i+1/2} = t_i + 0.5\Delta t$ and use this velocity to compute the displacement at t_{i+1}. Using the displacement at t_{i+1}, we can compute the velocity at t_{i+1}. Algorithm is as follows:

$$\begin{aligned}\text{(velocity at mid-point)} \quad & v(t_{i+1/2}) = v(t_i) + 0.5\Delta t f(u(t_i), v(t_i)),\\ \text{(displacement at next point)} \quad & u(t_{i+1}) = u(t_i) + \Delta t v(t_{i+1/2}),\\ \text{(velocity at new point)} \quad & v(t_{i+1}) = v(t_{i+1/2}) + 0.5\Delta t f(u(t_{i+1}), v(t_{i+1/2})).\end{aligned} \tag{8}$$

2. Sampling from Gaussian distribution in Matlab. In Matlab, you can use `randn` function to sample from the *Normal* distribution $\mathcal{N}(0, 1)$. (Note that Normal distribution is a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ with mean $\mu = 0$ and standard deviation $\sigma = 1$.

In **Problem 1**, you need to get samples from Gaussian distribution $\mathcal{N}(0, \sigma)$ where $\sigma = 0.1$. You can do get one sample using `sigma = 0.1; x_sample = sigma*randn(1,1)`. Or you can extract all 10 samples in one go using `sigma = 0.1; x_samples = sigma*randn(10,1)`. To know more about `randn`, use Matlab `help randn`.

3. Linear regression and interpolation in Matlab Given a vector of points `x` and data values `y`, use `p = polyfit(x, y, n)` with `n = 10` (order of polynomial) to get the polynomial interpolation. Note that if the order of polynomial `n` is exactly equal to `length(x) - 1` then `polyfit` performs polynomial interpolation. And if `n < length(x) - 1` it performs the linear regression using n-th order polynomial.

You can evaluate the polynomial function at any points. Suppose `z` is the vector of points at which we want to evaluate polynomial function. We can write in Matlab `yz = polyval(p, z)`, where `p` is from `p = polyfit(x, y, n)`. You can plot the interpolated polynomial function by simply writing `plot(z, yz)`.

4. Runge's phenomenon. This problem shows that when interpolating a function, the choice of points x_i where interpolated function and actual function agree is very important. From the error analysis of interpolation, we know that error function has $n+1$ roots for n-th order polynomial interpolation and the location of roots of error function is exactly the selected points x_i.

For the case when you choose uniformly spaced points to perform polynomial interpolation, this problem shows that having higher order polynomial is not a good idea. You can see that from the plot. Although the interpolated function agrees with the actual function at discrete points x_i, it shows large errors at other points. Higher order polynomial tend to be more oscillatory and this particular function highlights this behavior.

For the case when you choose points x_i using Chebyshev points, the error function will still have $n+1$ roots at these Chebyshev points x_i. However, due to the way these points are selected, the error at points other than discrete points are never high.

5. Gauss quadrature method. Consider following integration and it's approximation

$$I[f] = \int_{-1}^{1} g(x)\mathrm{d}x \approx \sum_{i=1}^{n} c_i g(x_i), \tag{9}$$

where n is the number of quadrature points in the Gauss quadrature approximation, and c_i and x_i are the weight and location of i-th point for $i = 1, 2, ..., n$. From the Table 20.1 of the reference book, for the two-point Gauss quadrature method ($n = 2$), we have.

$$c_1 = c_2 = 1, \qquad x_1 = -\frac{1}{\sqrt{3}}, x_2 = \frac{1}{\sqrt{3}} \tag{10}$$

For the three-point Gauss quadrature method ($n = 3$), we have

$$c_1 = c_3 = 5/9, c_2 = 8/9, \qquad x_1 = -\sqrt{\frac{3}{5}}, x_2 = 0, x_3 = \sqrt{\frac{3}{5}}. \tag{11}$$

Now, for the integration such as below

$$I[f] = \int_{a}^{b} f(x)\mathrm{d}x, \tag{12}$$

you can use change in variable to convert it into the integration

$$I[g] = \int_{-1}^{1} g(x)\mathrm{d}x. \tag{13}$$

This is done in the class. To summarize, let $y(x) = \alpha + \beta(x - a)$. We want $y = -1$ when $x = a$ and $y = 1$ when $x = b$. This gives us two equations to solve for α and β:

$$\begin{aligned} y(a) &= \alpha + \beta 0 = -1, \\ y(b) &= \alpha + \beta(b - a) = 1. \end{aligned} \tag{14}$$

Which results in $\alpha = -1$ and $\beta = 2/(b-a)$. Thus, we have $y(x) = -1 + 2(x-a)/(b-a)$. (Note that I used Newton interpolation to fit the two paired-data $(a, -1)$ and $(b, 1)$ by a straight-line $y = \alpha + \beta(x - a)$.)

Using $y = -1 + 2(x - a)/(b - a)$, we have $x = (b - a)(y + 1)/2 + a$ and $\mathrm{d}x = (b - a)\mathrm{d}y/2$. With change in variable from x to y, we have

$$I[f] = \int_{-1}^{1} f(a + (b - a)(y + 1)/2)(b - a)\mathrm{d}y/2 = \int_{-1}^{1} g(y)\mathrm{d}y = I[g], \tag{15}$$

where $g = g(y) = (b - a)f(a + (b - a)(y + 1))/2$. We know how to approximate $I[g]$ from (9).