Санкт-Петербургский государственный университет
Математико-механический факультет

Отчёт по практическому заданию №1

Метод конечных элементов для обыкновенного дифференциального уравнения второго порядка. Метод прогонки, Якоби и декомпозиции области.

Выполнил:
студент 21.Б12-мм гр.
Челабов А.Г.

Санкт - Петербург
2024

1.Поставновка задачи

Дана краевая задача $\quad -(a(x)u')' + c(x)u = f(x) \quad u(0) = u(1) = 0$

Это уравнение эквивалентно нахождению решения интегрального тождества: найти $u(x) \in H_0^1(0,1)$ такую, что

$$\int_0^1 (a(x)u'v' + c(x)uv)dx = \int_0^1 f(x)vdx, \qquad \forall v \in H_0^1\,(0,1)$$

Здесь

$$H_0^1(0,1) = \left\{v\colon \int_0^1 [(v')^2 + v^2]dx < \infty\,, v(0) = v(1) = 0\right\}$$

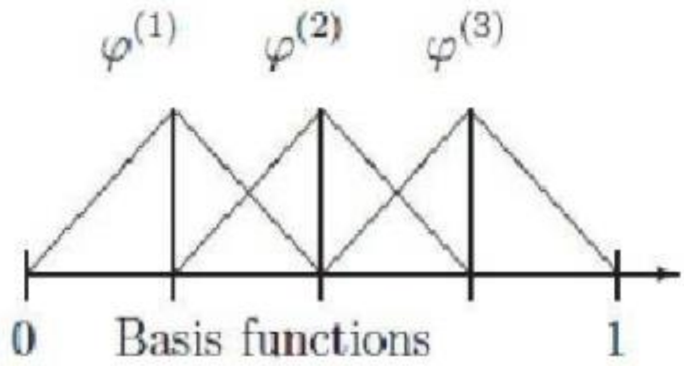
Введём сетку из N узлов и обозначим их x^i ($i = 0,1,\dots,N$, $x^{(0)} = 0,\ x^{(N)} = 1$). Приближённое решение будем искать в пространстве линейно-непрерывных функций $V_h^0(0,1) \subset H_0^1(0,1)$ на каждом конечном элементе $\tau_i = (x^{i-1}, x^i)$.

$$V_h^0(0,1) = \left\{v : v \in C(0,1),\ v\big|_{\tau_i} \in \mathcal{P}_1\,, i = 1,2,\dots,N;\ v(0) = v(1) = 0\right\}$$

Здесь $\mathcal{P}_1$ пространство полиномов первой степени.

Выберем в пространстве $V_h^0(0,1)$ базис $\{\varphi_i(x)\}_{i=1}^{N-1}$, определяемый равенствами

$$\phi_i(x^j) = \delta_j^i, i,j = 1,2,\dots,N-1$$



и запишем интегральное тождество, разложив v по этому базису:

$$\int_0^1 (au'\phi_i{}'(x) + cu\phi_i(x))dx = \int_0^1 f(x)\phi_i(x)dx$$

Решение ищется в виде $u = \sum_0^{N-1} \phi_i(x)u^i$, откуда следует, что u^i – приближенные значения в узлах x^i точного решения.

Подставив решение в интегральное тождество приближённое решение получим систему алгебраических уравнений:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$$

где $\boldsymbol{u} = \left\{u^i\right\}_{i=1}^{N-1}$ – вектор приближённых решений, а

$\boldsymbol{K} = \left\{k_{i,j}\right\}_{i=1}^{N-1} \quad k_{i,j} = \int_0^1 [a(x)\phi_i{}'\phi_j' + c(x)\phi_i\phi_j]dx$

$\boldsymbol{f} = \{f_i\}_{i=1}^{N-1} \qquad f_i = \int_0^1 f(x)\phi_i dx$

Матрица $\boldsymbol{K}$ – трёхдиагональная. Решим эту систему следующими методами.

2.1. Метод прогонки

Система имеет вид

$$k_{i,i-1}u^{i-1} + k_{i,i}u^i + k_{i,i+1}u^{i+1} = f_i$$

Ищем решение системы в виде $u^i = s_i u^{i+1} + t_i \quad i = 1, 2, \dots, N-1$

Т.к. матрица трёхдиагональная для $i = 1$ получаем

$$k_{1,1}u^1 + k_{1,2}u^2 = f_1$$

Перепишем

$$u^1 = -\frac{k_{1,2}}{k_{1,1}}u^2 + \frac{f_1}{k_{1,1}}$$

Таким образом

$$s_1 = -\frac{k_{1,2}}{k_{1,1}}$$

$$t_1 = \frac{f_1}{k_{1,1}}$$

Подставим u^{i-1} в i-тое уравнение системы

$$k_{i,i-1}\left(s_{i-1}u^i + t_{i-1}\right) + k_{i,i}u^i + k_{i,i+1}u^{i+1} = f_i$$

Преобразуем

$$u^i = -\frac{k_{i,i+1}}{k_{i,i-1}s_{i-1} + k_{i,i}}u^{i+1} + \frac{f_i - t_{i-1}k_{i,i-1}}{k_{i,i-1}s_{i-1} + k_{i,i}}$$

Таким образом

$$s_i = -\frac{k_{i,i+1}}{k_{i,i-1}s_{i-1} + k_{i,i}}$$
$$t_i = \frac{f_i - t_{i-1}k_{i,i-1}}{k_{i,i-1}s_{i-1} + k_{i,i}}$$

Для $t_{N-1} = u^{N-1}$

Делаем обратную прогонку, подставляя полученные коэффициенты, в

$$u^i = s_i u^{i+1} + t_i$$

2.2. Метод Якоби

Итерационный алгоритм с параметром σ. Для нашей системы алгебраических уравнений он выглядит так:

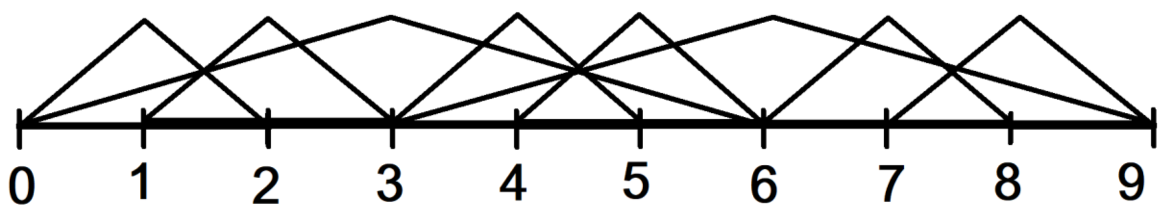
$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k - \sigma \boldsymbol{D}^{-1}(\boldsymbol{K}\boldsymbol{u}^k - \boldsymbol{f})$$

где $\boldsymbol{D}$ − диагональная матрица, совпадающая на главной диагонали с $\boldsymbol{K}$.

2.3. Метод декомпозиции области

Пусть $n = \sqrt{N}$. Введём новый базис $\{\psi\}_{i=1}^{N-1}$ в $V_h(0,1)$.

$\psi_i = \phi_i$ для $i \in \omega_r = \{(r-1)n + 1, (r-1)n + 2, \dots, rn - 1)\}$. Для оставшихся базисных векторов введём новую сетку $z^r = x^{nr}$ с шагом $H = nh$ и определим их аналогично предыдущему базису. Получаем ещё $\left\{\phi_{H,r}\right\}_{r=1}^{n-1}$.



Перенумеруем новый базис:

$$\psi_i(x) = \begin{cases} \phi_j(x), & i = 1,2,\dots,n(n-1), \qquad j = i + \left\lfloor \dfrac{i}{n-1} \right\rfloor \\ \phi_{H,r}(x), & i = n(n-1) + r, \qquad r = 1,2,\dots,n-1 \end{cases}$$

Система алгебраических уравнений в новом базисе имеет вид:

$$\boldsymbol{K}_{DD}\boldsymbol{u}_{DD} = \boldsymbol{f}_{DD}$$

Где $\boldsymbol{K}_{DD} = \left\{\boldsymbol{K}_{DD,r,p}\right\}_{r,p=1}^{n+1} -$ $(n+1)\times(n+1)$- блочно-структурная матрица, у которой отличны от нуля только блоки на диагонали и блоки у которых, один из индексов r, p равен $n+1$.

Рассмотрим матрицу перехода $\boldsymbol{C}$ от старого базиса к новому. Она имеет блочно-структурный вид, где ненулевые блоки находятся на главной диагонали и в $n-1$ последних столбцах. Блоки в последних столбцах будут иметь вид:

$$\begin{pmatrix} & & \begin{bmatrix} \frac{1}{n} & 0 & 0 \\ \dots & 0 & 0 \\ \frac{n-1}{n} & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} \dots & \cdots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix} & 0 & \begin{bmatrix} \frac{n-1}{n} & \frac{1}{n} & 0 \\ \dots & \dots & 0 \\ \frac{1}{n} & \frac{n-1}{n} & 0 \end{bmatrix} \\ & & \begin{bmatrix} 0 & \frac{n-1}{n} & \frac{1}{n} \\ 0 & \dots & \dots \\ 0 & \frac{1}{n} & \frac{n-1}{n} \end{bmatrix} \\ 0 & \ddots & \vdots \\ 0 & 0 & \begin{bmatrix} \dots & \dots & \dots \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix} \end{pmatrix}$$

Решив полученную новую систему, мы получим вектор $\boldsymbol{u}_{DD}$, который умножив на матрицу $\boldsymbol{C}$ получим нужный нам вектор $\boldsymbol{u}$. А матрица $\boldsymbol{K}_{DD}$ получается так:

$$\boldsymbol{K}_{DD} = \boldsymbol{C}^T\boldsymbol{K}\boldsymbol{C}$$

Систему с матрицей $\boldsymbol{K}_{DD}$ решаем итерационным методом:

$$\boldsymbol{u}_{DD}^{k+1} = \boldsymbol{u}_{DD}^{k} - \sigma \mathrm{B}^{-1}(\boldsymbol{K}_{DD}\boldsymbol{u}_{DD}^{k} - \boldsymbol{f}_{DD})$$

Где B – предобусловливатель. Можно без существенных потерь эффективности использовать более простые предобусловливатели. Если

$$0 < \mu_1 \le a(x) \le \mu_2$$

$$0 < c(x) \le \ \mu_1/2$$

и μ_1, μ_2 – постоянные, не сильно различающиеся, то можно принять $B = \Delta_{Hh}$

$$\mathbf{\Delta}_{Hh} = a_{\text{mean}}\text{diag}\left[\ \underbrace{\mathbf{\Delta}_h, \mathbf{\Delta}_h, .., \mathbf{\Delta}_h}_{n \text{ times}}, \mathbf{\Delta}_H\ \right]$$

a_{mean} – среднее значение на (0,1) коэффициента *a(x)* и Δ_h - $(n-1) \times (n-1)$ матрицы

$$\mathbf{\Delta}_h = \frac{1}{h}\begin{pmatrix} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & 0 & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & 0 & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{pmatrix}$$

Тогда решаем систему $\boldsymbol{u}_{DD}^{k+1} = \boldsymbol{u}_{DD}^k - \sigma \mathrm{B}^{-1}(\boldsymbol{K}_{DD}\boldsymbol{u}_{DD}^k - \boldsymbol{f}_{DD})$ в три этапа:

1) $\boldsymbol{d^k} = \ \boldsymbol{K}_{DD}\boldsymbol{u}_{DD}^k - \boldsymbol{f}_{DD}$
2) $\sigma \mathrm{B}^{-1}\boldsymbol{d^k} = \ \boldsymbol{w^k} \ \Leftrightarrow \mathrm{B}\boldsymbol{w^k} = \ \sigma\boldsymbol{d^k}$ (решаем методом **прогонки**)
3) $\boldsymbol{u}_{DD}^{k+1} = \boldsymbol{u}_{DD}^k - \ \boldsymbol{w^k}$

3.1. О выборе σ

Для определения σ, близкого к оптимальному

$$\sigma_{opt} \ = \frac{2}{\lambda_{min}(\boldsymbol{K}) + \ \lambda_{max}(\boldsymbol{K})}$$

Нужно прежде всего оценить $\lambda_{max}(\boldsymbol{K})$. Можно воспользоваться оценкой Фробениуса

$$\lambda_{max}(\boldsymbol{K}) \le max_j \sum_{i=1}^{N-1} |k_{i,j}|,$$

В результате которой убеждаемся, что

$$\lambda_{max}(\boldsymbol{K}) \le \ \mathbb{C}_2 h^{-1}, \ \ \mathbb{C}_2 = const$$

Не трудно проверить, что

$$\lambda_{min}(\boldsymbol{K}) \ge \ \mathbb{C}_1 h, \ \ \mathbb{C}_1 = const$$

Обратные оценки с теми же порядками получим, оценивая отношение Релея $\mathbf{v^T K v / v^T v}$ на подходящих фиксированных векторах $\boldsymbol{v}$. В совокупности они показывают, что при достаточно густой сетке $\lambda_{min}(\boldsymbol{K}) \ll \lambda_{max}(\boldsymbol{K})$, и можно использовать $\sigma = 2/\mathbb{C}_2 h^{-1}$. Для конечноэлементных матриц $\boldsymbol{K}$ оценка $\lambda_{min}(\boldsymbol{K}) \geq \mathbb{C}_1 h$ доказывается цепочкой неравенств

$$\mathbf{v}^{\mathrm{T}}\mathbf{K}\mathbf{v} \geq \mu_1(\mathrm{v}',\mathrm{v}') \geq \pi^2(v,v) = \pi^2\mathbf{v}^{\mathrm{T}}\mathbf{M}\mathbf{v} \geq \frac{\pi^2}{3}h\mathbf{v}^{\mathrm{T}}\mathbf{v} \quad \forall\mathbf{v} \Leftrightarrow v \in V_h^0(0,1)$$

где $\mathbf{M} = \{(\varphi_i,\varphi_j)\}_{i,j=1}^{N-1} = \mathrm{tridiag}\left[\frac{1}{6},\frac{2}{3},\frac{1}{6}\right]$ – матрица $N-1 \times N-1$, для которой $\lambda_{min}(\mathbf{M}) \geq \frac{1}{3}h$ и

$$(v,w) = \int_0^1 vwdx.$$

Реализация на языке Elixir

Вариант 7

$-y'' + r(x)\,y = f(x)$

$r(x)=e^{-x}, \quad f(x)=x-x^2$

```
defmodule NumericalMethods do
  import :math, only: [exp: 1, sqrt: 1]
  alias :lists, as: Lists

  # Вариант 7
  def p(_x), do: 1

  def r(x), do: exp(-x)

  def f(x), do: x - x * x

  def q(_x), do: 0

  # Метод прогонки
  def tridiagonal_matrix_algorithm(k, f, n) do
    start_time = :os.system_time(:millisecond)
    s = :lists.duplicate(n - 2, 0)
    t = :lists.duplicate(n - 1, 0)
    y = :lists.duplicate(n - 1, 0)

    {s, t, y} = initialize_tridiagonal(k, f, s, t, n)

    for i <- 1..(n - 3) do
      s = List.replace_at(s, i, -k[i][i + 1] / (k[i][i] + k[i][i - 1] * s[i - 1]))
    end

    for i <- 1..(n - 2) do
      t = List.replace_at(t, i, (f[i][0] - k[i][i - 1] * t[i - 1]) / (k[i][i] + k[i][i - 1] * s[i - 1]))
    end

    y = List.replace_at(y, n - 2, t[n - 2])

    for i <- (n - 3)..0 do
      y = List.replace_at(y, i, s[i] * y[i + 1] + t[i])
    end
```

```
  work_time = :os.system_time(:millisecond) - start_time
  {y, work_time}
end

defp initialize_tridiagonal(k, f, s, t, n) do
  s = List.replace_at(s, 0, -k[0][1] / k[0][0])
  t = List.replace_at(t, 0, f[0][0] / k[0][0])
  {s, t, :lists.duplicate(n - 1, 0)}
end

# Метод Якоби
def jacoby_method(k, g, err, max_num, n, sigma) do
  start_time = :os.system_time(:millisecond)
  u_new = :lists.duplicate(n - 1, 0)
  d = diagonal_inverse(k)
  u_old = :lists.duplicate(n - 1, 0)

  for _ <- 1..max_num do
    u_old = u_new
    u_new = Lists.zip_with(u_old, multiply(k, u_old) |> subtract(g), fn u, d -> u - sigma * d end)

    if norm(u_new, u_old) < err do
      break
    end
  end

  work_time = :os.system_time(:millisecond) - start_time
  {u_new, work_time}
end

defp diagonal_inverse(k) do
  Enum.map(k, fn row ->
    Enum.map(row, fn elem -> if elem != 0, do: 1 / elem, else: 0 end)
  end)
end
defp multiply(a, b) do
  for row <- a do
    for col <- transpose(b) do
      Enum.zip_with(row, col, &(&1 * &2)) |> Enum.sum()
    end
  end
end

defp subtract(a, b) do
  Enum.zip_with(a, b, &(&1 - &2))
end

defp norm(a, b) do
  a
  |> Enum.zip(b)
  |> Enum.map(fn {x, y} -> (x - y) * (x - y) end)
  |> Enum.sum()
  |> :math.sqrt()
end

defp transpose(matrix) do
  matrix
  |> Enum.zip()
  |> Enum.map(&Tuple.to_list/1)
end

# Матрицу коэффициентов
def get_coef_matrix(p, r, f, n) do
  h = 1 / n
  b = :lists.duplicate(n - 1, 0)
  a = :lists.duplicate(n - 2, 0)
  k = :lists.duplicate(n - 1, :lists.duplicate(n - 1, 0))
  f_matrix = :lists.duplicate(n - 1, [0])
```

```elixir
  for i <- 0..(n - 2) do
    f_matrix = List.replace_at(f_matrix, i, [f.(i * h + h) * h])
    b = List.replace_at(b, i, (2 / 3) * r.(i * h + h) * h + (p.(i * h + (0.5 * h)) + p.(i * h + (1.5 * h))) / h)
  end

  for i <- 0..(n - 3) do
    a = List.replace_at(a, i, (1 / 6) * r.(i * h + (1.5 * h)) * h - p.(i * h + (1.5 * h)) / h)
  end

  k = List.replace_at(k, 0, List.replace_at(k[0], 0, b))
  k = List.replace_at(k, 1, List.replace_at(k[1], 1, a))

  for i <- 1..(n - 2) do
    k = List.replace_at(k, i, List.replace_at(k[i], i, b))
    k = List.replace_at(k, i, List.replace_at(k[i], i - 1, a))
    k = List.replace_at(k, i, List.replace_at(k[i], i + 1, a))
  end

  {k, f_matrix}
end

# Получаем сигму
def get_sigma(k, n) do
  h = 1 / n
  sum_of_elem = :lists.duplicate(n - 1, 0)
  sum_of_elem = List.replace_at(sum_of_elem, 0, k[0][0] + k[1][0])

  for j <- 1..(n - 3) do
    sum_of_elem = List.replace_at(sum_of_elem, j, k[j][j] + k[j - 1][j] + k[j + 1][j])
  end

  sum_of_elem = List.replace_at(sum_of_elem, n - 2, k[n - 2][n - 2] + k[n - 3][n - 2])
  2 / Enum.max(sum_of_elem)
end
# Метод декомпозиции
def decomposition_method(k, f, err, max_num, n, sigma) do
  start_time = :os.system_time(:millisecond)
  m = trunc(sqrt(n))
  h = 1 / n

  # Формируем предобуславливатель
  delta_hh = :lists.duplicate(n - 1, :lists.duplicate(n - 1, 0))
  delta = :lists.duplicate(n - 1, :lists.duplicate(n - 1, 0))

  for i <- 0..(m * (m - 1) - 1) do
    delta = List.replace_at(delta, i, List.replace_at(delta[i], i, 2))
  end

  for i <- 0..(m * (m - 1) - 3) do
    delta = List.replace_at(delta, i, List.replace_at(delta[i], i + 1, -1))
    delta = List.replace_at(delta, i + 1, List.replace_at(delta[i + 1], i, -1))
  end

  delta = Enum.map(delta, fn row -> Enum.map(row, &(&1 / h)) end)

  for i <- (m * (m - 1))..(m * m - 2) do
    delta_hh = List.replace_at(delta_hh, i, List.replace_at(delta_hh[i], i, 2))
  end

  for i <- (m * (m - 1))..(m * m - 3) do
    delta_hh = List.replace_at(delta_hh, i, List.replace_at(delta_hh[i], i + 1,      -1))
    delta_hh = List.replace_at(delta_hh, i + 1, List.replace_at(delta_hh[i + 1], i, -1))
  end

  delta_hh = Enum.map(delta_hh, fn row -> Enum.map(row, &(&1 / (h * m))) end)
  delta_hh = Enum.zip_with(delta_hh, delta, fn a, b -> Enum.zip_with(a, b, &(&1 + &2)) end)

  p_arr = for i <- 0..100, do: p(i * (1 / 100))
```

```
  delta_hh = Enum.map(delta_hh, fn row -> Enum.map(row, &(&1 * p_mean)) end)

  # Матрица преобразования
  c = matrix_transform(n)
  kdd = multiply(transpose(c), multiply(k, c))
  fdd = multiply(transpose(c), f)

  u_new = :lists.duplicate(n - 1, 0)
  u_old = :lists.duplicate(n - 1, 0)
  counter = 0

  for _ <- 1..max_num do
    u_old = u_new
    counter = counter + 1
    d_k = multiply(kdd, u_old) |> subtract(fdd)

    # Применяем метод прогонки
    {w_k, _} = tridiagonal_matrix_algorithm(delta_hh, sigma * d_k, n)
    u_new = Lists.zip_with(u_old, w_k, &(&1 - &2))

    if norm(u_new, u_old) < err do
      break
    end
  end

  u_new = multiply(c, u_new)
  work_time = :os.system_time(:millisecond) - start_time
  {u_new, work_time, counter}
end

defp matrix_transform(n) do
  m = trunc(sqrt(n))
  c = :lists.duplicate(n - 1, :lists.duplicate(n - 1, 0))
  for i <- 0..(n - 2) do
    c = List.replace_at(c, i, List.replace_at(c[i], i, 1))
  end

  kn = 1
  for i <- 0..((m - 1) * (m - 1) - 1) do
    c = List.replace_at(c, i, List.replace_at(c[i], m * (m - 1) + div(i, (m - 1)), kn / m))
    kn = kn + 1
    if kn == m, do: kn = 1
  end

  kn = m - 1
  for i <- m - 1..((m - 1) * m - 1) do
    c = List.replace_at(c, i, List.replace_at(c[i], m * (m - 1) + div(i, (m - 1)) - 1, kn / m))
    kn = kn - 1
    if kn == 0, do: kn = m - 1
  end

  c
end

# Основная часть программы
def main do
  # Вызов функций
  n = IO.gets("Введите N: ") |> String.trim() |> String.to_integer()
  {k, f} = get_coef_matrix(&p/1, &r/1, &f/1, n)
  sigma = get_sigma(k, n)
  {y, work_time} = tridiagonal_matrix_algorithm(k, f, n)
  {y_j, work_time_j} = jacoby_method(k, f, 1e-4, 10000, n, sigma)

  xh = for i <- 1..(n - 1), do: i / n
```

```
# n_decomp - количество элементов в сетке для метода декомпозиции
n_decomp = 100
{k1, f1} = get_coef_matrix(&p/1, &r/1, &f/1, n_decomp)
{y_d, work_time_d, counter} = decomposition_method(k1, f1, 1e-4, 10000, n_decomp, sigma)
xh_d = for i <- 1..(n_decomp - 1), do: i / n_decomp

# Вывод таблицы значений
IO.puts("N = #{n}:")
IO.puts("Времена выполнения алгоритма:")
IO.puts("Прогонка: #{work_time} Якоби: #{work_time_j} Декомпозиция: #{work_time_d} сек\n")
IO.puts(String.pad_leading("x", 3) <> String.pad_leading("y(x)", 15) <>
  String.pad_leading("yJacoby(x)", 15) <> String.pad_leading("yDecomp(x)", 15))

for i <- 1..9 do
  IO.puts(
    String.pad_leading(Float.to_string(i * 0.1, decimals: 1), 2) <>
    String.pad_leading(Float.to_string(y[(n * i) // 10], decimals: 12), 15) <>
    String.pad_leading(Float.to_string(y_j[(n * i) // 10], decimals: 12), 15) <>
    String.pad_leading(Float.to_string(y_d[(n_decomp * i) // 10], decimals: 12), 15)
  )
end

IO.puts("\n")
IO.puts(String.pad_leading("№", 7) <> String.pad_leading("k1", 15) <>
  String.pad_leading("k2", 17) <> String.pad_leading("k3", 17) <> String.pad_leading("fi", 17))

for i <- 1..9 do
  IO.puts(
    String.pad_leading(Integer.to_string((n * i) // 10), 4) <>
    String.pad_leading(Float.to_string(k[(n * i) // 10][(n * i) // 10], decimals: 12), 16) <>
    String.pad_leading(Float.to_string(k[(n * i) // 10][((n * i) // 10) - 1], decimals: 12), 16) <>
    String.pad_leading(Float.to_string(k[(n * i) // 10][((n * i) // 10) + 1], decimals: 12), 16) <>
    Float.to_string(f[(n * i) // 10][0], decimals: 12)
  )
end
```

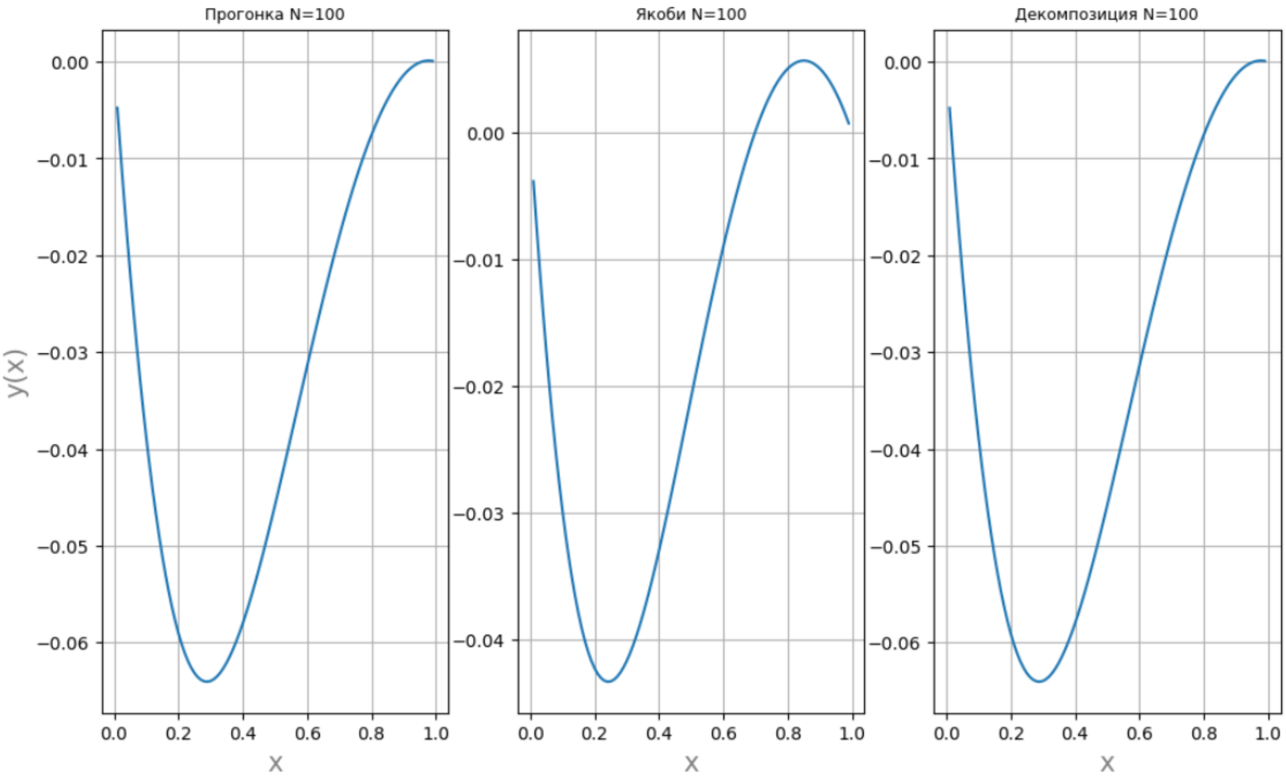
Результаты выполнения программы

а) N = 100

```
N =  100 :
Времена выполнения:
Прогонка:  0.0020003318786621094  Якоби:  0.21299958229064941  Декомпозиция:  0.007999897003173828  сек
```

x	U1(x)	U2(x)	U3(x)
0.1	-0.041638293081	-0.031724980679	-0.041638293081
0.2	-0.060174338741	-0.042676144152	-0.060174338741
0.3	-0.063805319444	-0.041068024568	-0.063805319444
0.4	-0.057028373609	-0.031863697975	-0.057028373609
0.5	-0.044443324998	-0.019633271401	-0.044443324998
0.6	-0.030031550922	-0.007944213644	-0.030031550922
0.7	-0.016801074802	0.000809597272	-0.016801074802
0.8	-0.006749109470	0.005260112408	-0.006749109470
0.9	-0.001019202496	0.004781362429	-0.001019202496

№	A	B	C	F
10	200.001466666667	-99.999650000000	-99.999616666667	-0.018590949881387472
20	200.002800000000	-99.999316666667	-99.999283333333	-0.015246610505719016
30	200.004133333333	-99.998983333333	-99.998950000000	-0.010808773799444834
40	200.005466666667	-99.998650000000	-99.998616666667	-0.006220264363758312
50	200.006800000000	-99.998316666667	-99.998283333333	-0.002196064127725901
60	200.008133333333	-99.997983333333	-99.997950000000	0.0009004351029682782
70	200.009466666667	-99.997650000000	-99.997616666667	0.0030110933363653625
80	200.010800000000	-99.997316666667	-99.997283333333	0.0042636373301838705
90	200.012133333333	-99.996983333333	-99.996950000000	0.0048590552490201605

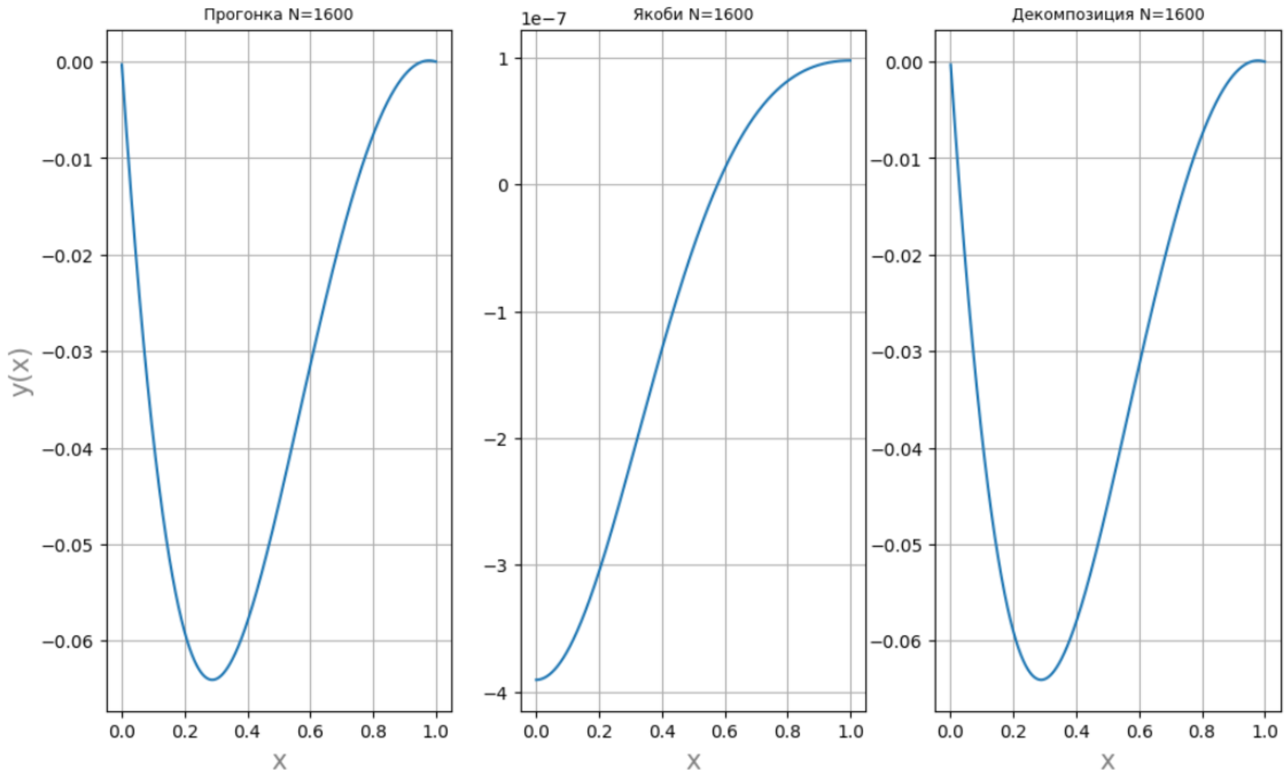


б) N = 1600

N = 1600 :
Времена выполнения:
Прогонка: 0.019999980926513672 Якоби: 0.003000020980834961 Декомпозиция: 0.21800732612609863 сек

x	U1(x)	U2(x)	U3(x)
0.1	-0.038998787894	-0.000000367483	-0.038998787894
0.2	-0.059133012342	-0.000000305111	-0.059133012342
0.3	-0.063980815588	-0.000000219626	-0.063980815588
0.4	-0.057981100743	-0.000000129590	-0.057981100743
0.5	-0.045759426395	-0.000000049550	-0.045759426395
0.6	-0.031375594712	0.000000012752	-0.031375594712
0.7	-0.017934482636	0.000000055713	-0.017934482636
0.8	-0.007523517399	0.000000081590	-0.007523517399
0.9	-0.001356941738	0.000000094249	-0.001356941738

№	A	B	C	F
160	3200.000083854167	-1599.999979101563	-1599.999978971354	-0.0011759461360086052
320	3200.000167187500	-1599.999958268229	-1599.999958138021	-0.0009763558854499137
480	3200.000250520834	-1599.999937434896	-1599.999937304688	-0.0007028040345740313
640	3200.000333854166	-1599.999916601563	-1599.999916471354	-0.0004146884306969091
800	3200.000417187500	-1599.999895768229	-1599.999895638021	-0.00015856114054869742
960	3200.000500520833	-1599.999874934896	-1599.999874804688	4.080531213753513e-05
1120	3200.000583854167	-1599.999854101563	-1599.999853971354	0.0001782804583878814
1280	3200.000667187500	-1599.999833268229	-1599.999833138021	0.0002610875795861688
1440	3200.000750520833	-1599.999812434896	-1599.999812304688	0.0003015953465238074

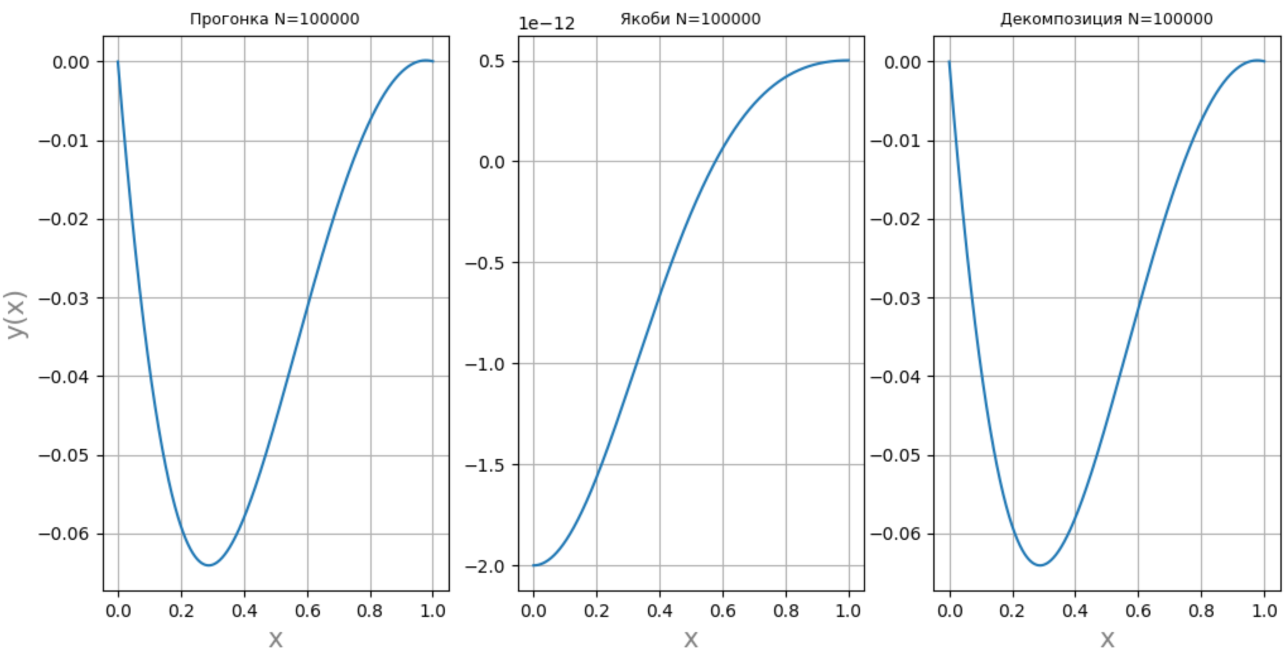


в) N = 10000

```
N =  10000 :
Времена выполнения:
Прогонка:  0.06599974632263184  Якоби:  0.026000022888183594  Декомпозиция:  16.404548406600952  сек
```

x	U1(x)	U2(x)	U3(x)
0.1	-0.038846193987	-0.000000000002	-0.038835215169
0.2	-0.059070771027	-0.000000000002	-0.059054076283
0.3	-0.063987817999	-0.000000000001	-0.063969733586
0.4	-0.058032756463	-0.000000000001	-0.058016355086
0.5	-0.045832421218	-0.000000000000	-0.045819467933
0.6	-0.031450925053	0.000000000000	-0.031442036306
0.7	-0.017998560521	0.000000000000	-0.017993473719
0.8	-0.007567843572	0.000000000000	-0.007565704728
0.9	-0.001377030667	0.000000000000	-0.001376641487

№	A	B	C	F
1000	20000.000013346667	-9999.999996664999	-9999.999996661667	-0.00018827164552453297
2000	20000.000026680002	-9999.999993331667	-9999.999993328332	-0.00015642396375382513
3000	20000.000040013332	-9999.999989998334	-9999.999989995000	-0.00011269237097482
4000	20000.000053346666	-9999.999986665000	-9999.999986661667	-6.658386399184117e-05
5000	20000.000066680001	-9999.999983331667	-9999.999983328333	-2.5563140669800318e-05
6000	20000.000080013335	-9999.999979998332	-9999.999979995000	6.387610056277753e-06
7000	20000.000093346665	-9999.999976665000	-9999.999976661667	2.8433778382154247e-05
8000	20000.000106680000	-9999.999973331667	-9999.999973328333	4.1723978220456874e-05
9000	20000.000120013334	-9999.999969998333	-9999.999969995000	4.823527031697251e-05

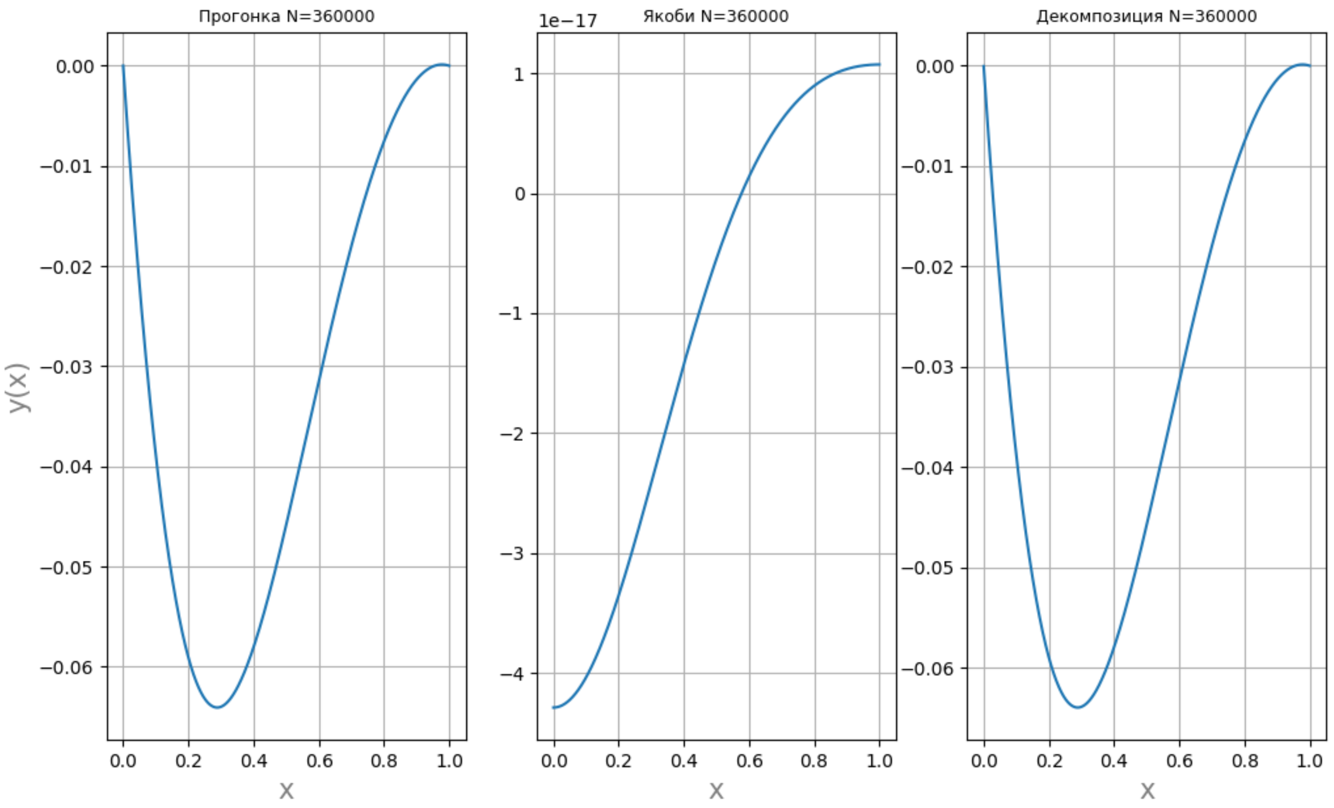


д) N = 360000

```
N =  360000 :
Времена выполнения:
Прогонка:  2.548297643661499  Якоби:  0.802067756652832  Декомпозиция:  1388.7399559020996  сек

 x       U1(x)           U2(x)           U3(x)
0.1 -0.038817879964 -0.000000000000 -0.038798725474
0.2 -0.059059198849 -0.000000000000 -0.058998588879
0.3 -0.063989081559 -0.000000000000 -0.063909627415
0.4 -0.058042302292 -0.000000000000 -0.057961842728
0.5 -0.045845930293 -0.000000000000 -0.045776415811
0.6 -0.031464875646  0.000000000000 -0.031412493266
0.7 -0.018010433687  0.000000000000 -0.017976566992
0.8 -0.007576063144  0.000000000000 -0.007558595967
0.9 -0.001380764393  0.000000000000 -0.001375347990
```

```
   №            A                   B                   C                  F
36000 720000.000000370434 -359999.999999907392 -359999.999999907392  -5.230384845638956e-06
72000 720000.000000740751 -359999.999999814841 -359999.999999814783  -4.3461740270024826e-06
108000 720000.000001111068 -359999.999999722233 -359999.999999722233  -3.1315971876162705e-06
144000 720000.000001481501 -359999.999999629625 -359999.999999629625  -1.85075446995339e-06
180000 720000.000001851819 -359999.999999537016 -359999.999999537016  -7.11082666766749e-07
216000 720000.000002222252 -359999.999999444466 -359999.999999444466  1.767062241417475e-07
252000 720000.000002592569 -359999.999999351858 -359999.999999351858  7.89357840967803e-07
288000 720000.000002963003 -359999.999999259249 -359999.999999259249  1.1587414465284006e-06
324000 720000.000003333320 -359999.999999166641 -359999.999999166641  1.3397654196937995e-06
```

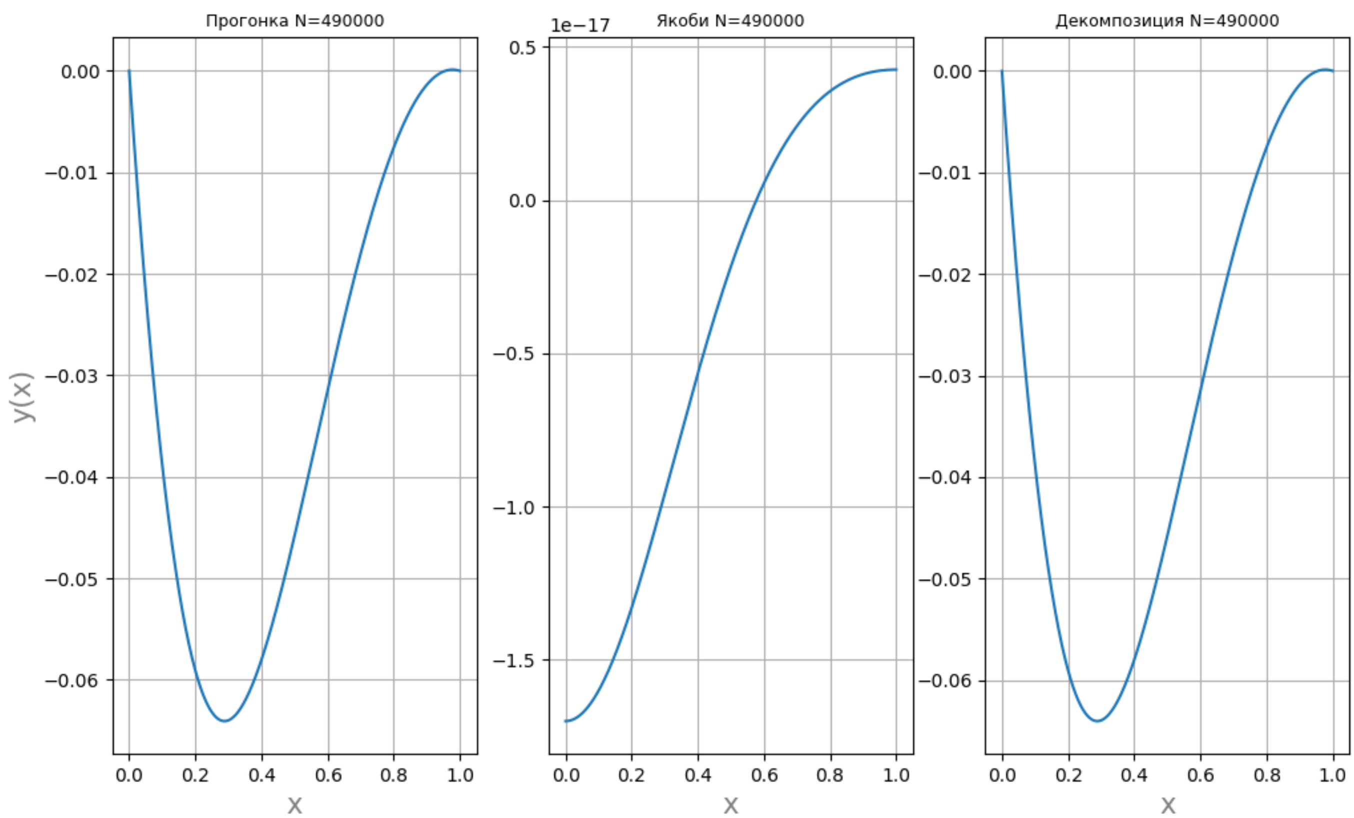


e) N = 490000

```
N =  490000 :
Времена выполнения:
Прогонка:  3.3092238903045654  Якоби:  1.0790393352508545  Декомпозиция:  2719.7285470962524  сек

 x       U1(x)           U2(x)           U3(x)
0.1 -0.038817665248 -0.000000000000 -0.038818987743
0.2 -0.059059111039 -0.000000000000 -0.059047384038
0.3 -0.063989091016 -0.000000000000 -0.063970366534
0.4 -0.058042374538 -0.000000000000 -0.058021727308
0.5 -0.045846032609 -0.000000000000 -0.045827120970
0.6 -0.031464981339  0.000000000000 -0.031449961483
0.7 -0.018010523658  0.000000000000 -0.018000230839
0.8 -0.007576125442  0.000000000000 -0.007570389208
0.9 -0.001380792702  0.000000000000 -0.001378771979


   №            A                   B                     C                    F
49000 980000.000000272179 -489999.999999932013 -489999.999999932013  -3.842735157474958e-06
98000 980000.000000544358 -489999.999999864027 -489999.999999864027  -3.193113372627432 7e-06
147000 980000.000000816421 -489999.999999795982 -489999.999999795982  -2.3007722616887396e-06
196000 980000.000001088600 -489999.999999727937 -489999.999999727937  -1.3597446762464073e-06
245000 980000.000001360662 -489999.999999659893 -489999.999999659893  -5.22433626028861e-07
294000 980000.000001632725 -489999.999999591906 -489999.999999591906  1.2982092935498125e-07
343000 980000.000001904904 -489999.999999523861 -489999.999999523861  5.799337585214379e-07
392000 980000.000002176967 -489999.999999455817 -489999.999999455817  8.513188093865599e-07
441000 980000.000002449146 -489999.999999387830 -489999.999999387830  9.84316876122856e-07
```



ж) N = 810000

```
N =  810000 :
Времена выполнения:
Прогонка:  6.35846471786499  Якоби:  3.239141941070556  Декомпозиция:  6842.14803314209  сек
```

x	U1(x)	U2(x)	U3(x)
0.1	-0.038817430507	-0.000000000000	-0.038817301027
0.2	-0.059059015261	-0.000000000000	-0.059046561486
0.3	-0.063989101754	-0.000000000000	-0.063970239169
0.4	-0.058042453977	-0.000000000000	-0.058022076657
0.5	-0.045846144855	-0.000000000000	-0.045827727270
0.6	-0.031465097184	0.000000000000	-0.031450636560
0.7	-0.018010622251	0.000000000000	-0.018000832428
0.8	-0.007576193702	0.000000000000	-0.007570820767
0.9	-0.001380823725	0.000000000000	-0.001378974380

№	A	B	C	F
81000	1620000.000000164611	-809999.999999958905	-809999.999999958789	-2.3246198371460276e-06
162000	1620000.000000329223	-809999.999999917694	-809999.999999917694	-1.9316404053896366e-06
243000	1620000.000000493834	-809999.999999876600	-809999.999999876600	-1.3918298154502042e-06
324000	1620000.000000658445	-809999.999999835389	-809999.999999835389	-8.225660273116133e-07
405000	1620000.000000823056	-809999.999999794294	-809999.999999794294	-3.16043764069143e-07
486000	1620000.000000987668	-809999.999999753083	-809999.999999753083	7.853096718245138e-08
567000	1620000.000001152279	-809999.999999711988	-809999.999999711988	3.5082239525229694e-07
648000	1620000.000001316890	-809999.999999670777	-809999.999999670777	5.149943780767577e-07
729000	1620000.000001481501	-809999.999999629683	-809999.999999629683	5.954505691539007e-07